Chapter 4: Overview of Traditional and Recent Heuristic Optimization Methods

Chapter details

Chapter DOI:

https://doi.org/10.4322/978-65-86503-71-5.c04

Chapter suggested citation / reference style:

Choze, Sergio B., et al. (2022). "Overview of Traditional and Recent Heuristic Optimization Methods". *In* Jorge, Ariosto B., et al. (Eds.) *Model-Based and Signal-Based Inverse Methods*, Vol. I, UnB, Brasilia, DF, Brazil, pp. 107–142. Book series in Discrete Models, Inverse Methods, & Uncertainty Modeling in Structural Integrity.

P.S.: DOI may be included at the end of citation, for completeness.

Book details

Book: Model-based and Signal-Based Inverse Methods

Edited by: Jorge, Ariosto B., Anflor, Carla T. M., Gomes, Guilherme F., & Carneiro, Sergio H. S.

Volume I of Book Series in:

Discrete Models, Inverse Methods, & Uncertainty Modeling in Structural Integrity

Published by: UnB City: Brasilia, DF, Brazil Year: 2022

DOI: https://doi.org/10.4322/978-65-86503-71-5

Overview of Traditional and Recent Heuristic Optimization Methods

Sergio B. Choze[1*] Rogerio R. Santos[2] Guilherme F. Gomes[3]

[1]Consultant. E-mail: sergio.butkewitsch@gmail.com
[2]Division of Mechanical Engineering, Technological Institute of Aeronautics - ITA, Brazil. E-mail: rsantos9@gmail.com
[3]Mechanical Engineering Institute, Federal University of Itajuba - UNIFEI, Brazil. E-mail: guilhermefergom@unifei.edu.br

*Corresponding author

Abstract

This chapter describes theoretical and practical aspects of Heuristic Optimization Methods. The introductory section describes the basic equations commonly found in numerical optimization and enumerates some open questions in the field. Next, general principles are outlined, and an overview of some heuristic optimization methods is presented, followed by considerations about the main aspects of numerical computations. The rest of the text is devoted to the clarification of method-specific variations and novel methods. Algorithms and fragments of source code are showed to enrich the discussion about the methodologies. Some practical considerations for applied optimization are the concluding remarks.

1 Introduction

Heuristic optimization methods are meant to handle special conditions of the same fundamental mathematical optimization problem introduced in the Chapter "Overview of Linear and Non-linear Programming Methods for Structural Optimization", as defined in Equations (1) - (4) below. This means that while the problem statement or formulation remains unchanged, the solution methods no longer rely on derivatives, either because 1) they are not available at all, or 2) because they are not effective to determine a search path leading to a satisfactory solution.

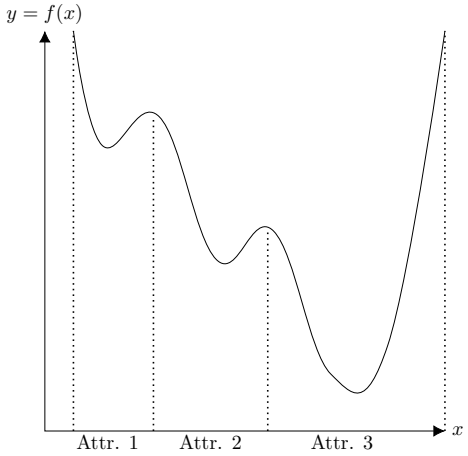
$$\min, \max F(\mathbf{x}) \tag{1}$$

subject to:

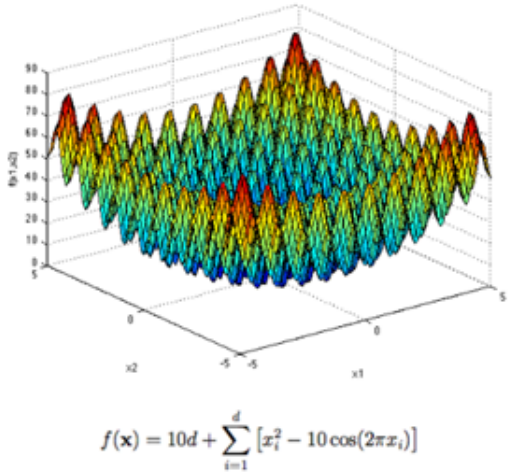
$$G_j(\mathbf{x}) \leq 0 \tag{2}$$

$$H_k(\mathbf{x}) = 0 \tag{3}$$

$$\mathbf{x}_{LB} \leq \mathbf{x} \leq \mathbf{x}_{UB} \tag{4}$$



(a) Three basins of attraction seen across one dimension.



(b) 2-dimensional case of the Rastrigin function, showing multiple basins of attraction.

Figure 1: Function with multiple basins of attraction.

Situation 1 is prone to arise in various circumstances, even when the optimization process is driven by actual physical experimentations that cannot be undertaken, but in the context of computational mechanics are usually associated with non-linear and discontinuous analyses in which convergence of a certain simulation scenario is not attainable, and hence information for calculation of derivatives is missing.

More often though, still in the realm of highly complex and non-linear search spaces, situation 2 happens when derivative based methods may be uncapable of navigating throughout the decision space at large, being confined to a narrow search domain which does not necessarily contains the best possible decision, which at this point is unknown or undetermined. The narrow sections of the overall decision space are called basins of attraction, which may contain local or global optima, as represented in Figure 1. Panel (a) sketches a simplified (for illustration purposes) one-dimensional function (or a one-dimensional cut of a multivariate function) whereby 3 adjacent basins of attraction are identified. Panel (b) shows the 2-dimensional case of the Rastrigin function [1], a common test function for the performance of the methods to be covered in this chapter. The functions have multiple basins of attraction: (a) schematic decision space with basins of attraction 1, 2, and 3, with the latter one containing a global optimum (function minimum) concerning the other two and (b) 2-dimensional Rastrigin function, with multiple basins of attraction and known analytical global optimum at (0, 0).

Many problems in engineering can be formulated as optimization problems, subject to complex nonlinear objective functions and constraints. As already stated by Yang (2020), the solutions of highly nonlinear problems usually require sophisticated optimization algorithms, and traditional algorithms may struggle to deal with such problems.

Many real-world engineering applications involve the optimization of certain objectives such as the minimization of costs, energy consumption, environment and the maximization of performance, efficiency and sustainability (Yang [2020]). In many

cases, the optimization problems that can be formulated are highly nonlinear with multimodal objective landscapes, subject to a set of complex, nonlinear constraints. Such problems are challenging to solve. Even with the ever-increasing power of modern computers, it is still impractical and not desirable to use simple brute force approaches. Thus, whenever possible, efficient algorithms are crucially important to such applications. However, efficient algorithms may not exist for most of the optimization problems in applications. Though there are a wide range of optimization algorithms such as gradient-based algorithms, the interior-point method and trust-region method, most of such algorithms are gradient-based and local search algorithms, which means that the final solutions may depend on the initial starting points. In addition, the computation of derivatives can be computationally expensive, and some problems such as the objective with discontinuities may not have derivatives in certain regions.

Despite the effectiveness of nature-inspired algorithms and their popularity, there are still many challenging issues concerning such algorithms, especially from theoretical perspectives.

The paper Yang [2020] highlights five main challenging issues that justify the use of advanced metaheuristic techniques. However, much remains to be explored in this field of research. The following stand out:

- Open Problem 1: How to build a unified framework for analyzing all nature-inspired algorithms mathematically, so as to obtain in-depth information about their convergence, rate of convergence, stability, and robustness?
- Open Problem 2: How to best tune the parameters of a given algorithm so that it can achieve its best performance for a given set of problems? How to vary or control these parameters so as to maximize the performance of an algorithm?
- Open Problem 3: What types of benchmarking are useful? Do free lunches exist, under what conditions?
- Open Problem 4: What are the most suitable performance metrics for fairly comparing all algorithms? Is it possible to design a unified framework to compare all algorithms fairly and rigorously?
- Open Problem 5: How to best scale up the algorithms that work well for small-scale problems to solve truly large-scale, real-world problems efficiently?

The authors of Yang [2020] concludes that there are other open problems concerning nature-inspired algorithms, including how to achieve the optimal balance of exploitation and exploration, how to deal with nonlinear constraints effectively, and how to use these algorithms for machine learning and deep learning. Nature-inspired computation is an active area of research. It is hoped that the above five open problems we have just highlighted can inspire more research in this area in the near future.

As indicated by the nomenclature, basins of attraction will draw the search method towards themselves, which is neither harmful nor advantageous per se, depending primarily on: 1) how sooner or later the attraction happens in the search

process and 2) how much decision making elements are available to ascertain that the optimum contained in a basin of attraction is the best decision, given a search cost requirement and 3) what are the resources available to afford a larger or smaller search effort, without explicit guarantee that a better solution is available at a different basin of attraction.

Despite of these considerations, a common feature of all basins of attraction is that they tend to contain derivative values of zero for functions that represent the decision spaces where the search is being performed, which means the stalling of classical, derivative driven optimization methods. In optimization jargon, this phenomenon is known as "being trapped at a local optimum" , and the goal of this chapter is to discuss the so-called heuristic optimization methods, which are meant to avoid this phenomenon by being more flexible than strict derivatives to balance exploration (mapping of a broader decision space, identifying as many as possible basins of attraction) with exploitation (deciding upon the best solution to be exploited, conditioned on the knowledge of the decision space accrued thus far by exploration and the budget to further the search). This flexibility is achieved by replacing the point-to-point search strategies characteristic of derivative driven methods by population-based approaches, which signify an additional cost to perform the search.

2 General principles

Heuristic optimization methods depart from multiple starting points, on the vein of balancing the odds across multiple potential (and often unknown, at least a priori) basins of attraction. Hence, they are from the onset population based, and balance should be achieved between how representative the population is (typically an artifact of its size, barring any biases) and how feasible it is to handle the computational cost of evaluating larger populations as they grow.

Once the multiple starting points are determined, instead of proceeding with derivative based searches for each of every of them (the multi-start concept in classical optimization (De Jong [2016])), which would ensue a totally deterministic operation of the methods thereafter, aleatory components are actually introduced to further balance the odds with respect to finding or avoiding the possible basins of attraction. On this vein, the values of the decision variables are randomly perturbed according to an approach particular to each method, which also rank the resulting perturbed decision points (candidate optima) based on their own particular criteria. Rather than continuing to randomly perturb the values of the decision variables, the design candidates arising from these perturbations are somehow combined or mixed (again, according to each method?s particularities) so that the effects of randomness apply population-wise, rather than just at each individual setup. The combined candidate designs are then appraised and, for the most part, those with superior performance will comprise a larger fraction of the candidates pool for the next iteration, with a minority of relatively underperforming individuals still kept for diversity of the overall population and feasibility of reaching to alternative basins of attraction. After multiple repetitions of this sample $\rightarrow$ perturb $\rightarrow$ combine $\rightarrow$ appraise $\rightarrow$ resample sequence, it is expected that deterministic and aleatory search compo-

nents are ideally balanced and, between the extremes of classical and totally random searches, the best tradeoff has been met and the most suitable design, within the best relative basin of attraction, has been achieved at the minimum computational cost.

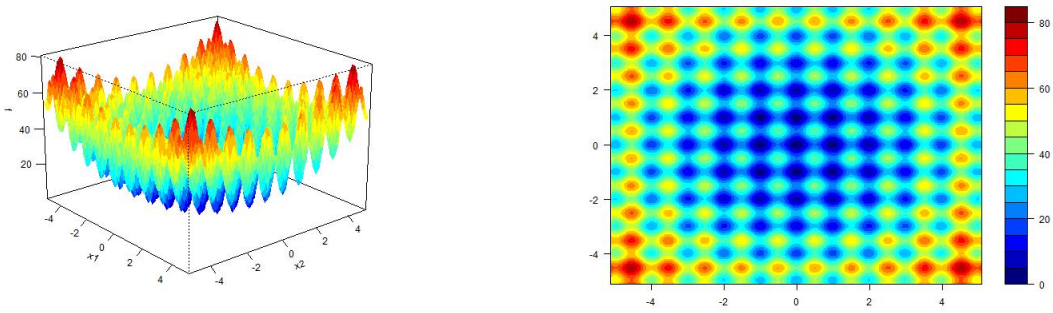
It should be noted that, along these lines, no formal convergence criteria similar to the Karush-Kuhn-Tucker (KKT) conditions apply, and the most adequate nomenclature is actually search termination, either because the allotted budget has been exhausted and/or the balance between random and deterministic search elements is such that no plausible improvement can be achieved. In order to strike this best balance even in the absence of formal criteria, intense research is devoted to the understanding and characterization of heuristic search methods as random processes, that may be modeled according to one of the canonical forms or combinations thereof available in this field of knowledge.

Lastly, since the aleatory-deterministic search balance effects one given function (typically the optimization objective expressed in Equation (1)), it is common not to account explicitly for constraint functions (equality or inequality, as in Equations (2) and (3)) for the execution of heuristic optimization methods. Because the decision making is not practical without any form of consideration of the constraints, they are either verified offline (simplistic approach) or embedded into the objective by way of a penalty (similarly to the constrained optimization methods detailed in Chapter "Overview of Linear and Non-linear Programming Methods for Structural Optimization", section 8) or, in the most sophisticated approaches, the problem statement is completely reconfigured as a multicriteria (Chapter "Overview of Linear and Non-linear Programming Methods for Structural Optimization", section 3) or multidisciplinary (Chapter "Overview of Linear and Non-linear Programming Methods for Structural Optimization", section 4) procedure.

Table 1 summarizes how the general principles outlined above are handled by each of the established search methods. Since these common principles are shared, the methods are inevitably similar to each other in many aspects, with the different approaches and respective nomenclatures arising more as an artifact of successful practice rather than formal mathematical features. The forthcoming sections are dedicated to explore each of the methods listed in Table 1 in greater detail.

Table 1: Overview of heuristic optimization methods according to the elements they use to balance aleatory and deterministic search components in pursuit of (potentially) global optima.

Method	Individual Unit	Population Unit	Iteration Unit	Operators	Comments
Genetic Algorithm	chromosomes, as a collection of genes (codified decision variables)	Population comprised of a determined number of individuals, each one with its chromosome	Generations, which get updated by successive application of the genetic operators	Selection, Crossover, Mutation	Basis (both historically and algorithmically) for most of the other heuristic methods. Multiple variations exist, but the core consists of applying the genetic operators in the order they appear in the previous column
Evolutionary Program	chromosomes, as a collection of genes (codified decision variables)	Population comprised of a determined number of individuals, each one with its chromosome	Generations, which get updated by successive application of the genetic operators	Mostly random mutations, at various categories and intensities	Similar to GAs, emphasizing the randomized element of the search through various modalities of mutations
Simulated Annealing	Candidate designs, as cohesive sets of decision variables	Sets of candidate designs	Annealing cycle	Heating, Cooling	
Ant Colony	Path traveled by individual ants	Path traveled by all ants	Excursions from and back to the ant colony, through a random path	Pheromone deposition and evaporation	
Particle Swarm	Particles	Swarm of many particles	Particle's Position and Velocity variations	Inertia, individual best and social best	
Differential Evolution	chromosomes, as a collection of genes (codified decision variables)	Population comprised of a determined number of individuals, each one with its chromosome	Generations, which get updated by successive application of the genetic operators	Mutation, Selection, Crossover	Same basic operators as GAs, but with mutation taking place earlier at each perturbation cycle



(a) (b)

Figure 2: The (a) Rastrigin test function and (b) the contour plot.

3 Numerical Computation

As outlined in Equation (1), the maximization and minimization of functions are similar from the computational perspective. Several algorithms are implemented aiming minimization. However, it is worth to note the difference in computational implementation that will switch between max and min. This feature will be used on snippet of computational codes along the text, implemented within the R statistical computation platform (R Core Team [2020]).

For this purpose, the reader may please refer to the command `install.packages` as the main tool to add new packages to the R software. It takes a vector of names and a destination library, downloads the packages from the repositories (from a local folder or remote web server) and installs them in the R environment. To install the packages needed to run the codes in this chapter, for example, enter the following

```
install.packages('GA', 'ecr', 'DEoptim', 'GenSA', 'pso',
  'metaheuristicOpt')
```

Help is available at the command prompt via `?install.packages` command.

As an example, consider the Rastrigin function, which is provided as test function aiming minimization. The global minimum value is 0 at the design $x = (0, .., 0)$. It is shown in Figure 2.

A two-dimensional implementation for minimization algorithms is given in Code 1.

Code 1: R implementation defining the 2D version of the Rastrigin function as the unconstrained objective to be minimized by various heuristic optimization methods.

```
obj2min <- function(x)
{
  obj=20+x[1]^2+x[2]^2-10*(cos(2*pi*x[1])+cos(2*pi*x[2]))
  return (obj)
}
```

On the other hand, a two-dimensional implementation aiming *maximization* algorithms is shown in Code 2.

Code 2: R implementation defining the 2D version of the Rastrigin function as the unconstrained objective to be maximized by various heuristic optimization methods.

```
obj2max <- function(x)
{
  obj=20+x[1]^2+x[2]^2-10*(cos(2*pi*x[1])+cos(2*pi*x[2]))
  return (-obj)
}
```

That is, the objective `obj2max` evaluated by a maximization algorithm will lead to the same result as the objective `obj2min` evaluated by the minimization algorithm. As a result, the multiplication of the objective function by -1 is sufficient to translate an unconstrained optimization problem between the minimization and maximization perspectives.

Following this template, the corresponding code snippets are offered the methods described throughout sections 4.1 to 4.6 below, as well as sections 5.3 and 5.4. These exemplary codes are offered whenever actual R packages (basic R topic modules) are available at the time of writing and able to offer implementations for each of the methods alluded within the aforementioned sections (group 4 for the more established techniques, and group 5 for the more recent developments).

The reader should notice minor variations in formatting and structure, due to particularities of each package implementing the applicable functions. At the same time, in an effort to create a multipurpose heuristic methods engine within the R statistical computing platform, Riza et al. [2019] proposed the `metaheuristicOpt` package, in which any of the specific algorithms listed (in alphabetical order) within Code 1 can be instantiated as determined by the parameter "`algorithm`" (Table 2) illustrated at the associated code fragment. The length of the list indicates the level of research activity in the field, as well as the commonality of nature inspired analogies underpinning the proposition of different heuristic methods that share several common principles. While the entire field is evolving, several novel methods introduce additional tuning parameters that may require a significant level of experimentation for fine tuning, and hence the most complex option is not necessarily the one yielding superior performance.

4 Method specific variations

4.1 Genetic Algorithms

As the very foundational of heuristic optimization methods, Genetic Algorithms balance random and deterministic decision-making efforts by way of the following sequence of operations:

- An encoding method, or schema, is adopted to codify combinations of decision variables in a string (per the biological analogy, a chromosome) that lead to

Code 3: Code fragment for minimization of the Rastrigin function through the metaheuristicOpt R package.

```
# Input #
obj2min <- function(x)
{
  return( 20 + x[1]^2 + x[2]^2 - 10 * (cos(2*pi*x[1]) +
    cos(2*pi*x[2]) ) )
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)

library('metaheuristicOpt')
BA_ans = metaOpt(obj2min, optimType='MIN', algorithm='BA',
  numVar = 2, rangeVar = rbind(lower_bound, upper_bound),
  control = list(numPopulation = 20, maxIter = 700) )
cat('\nOptimal␣value␣', BA_ans$optimumValue,
  '\nOptimal␣design␣', BA_ans$result)

# Output #
Optimal value  0
Optimal design  3.238586e-292 1.289825e-29
```

Table 2: Collection of heuristic optimization methods available through the metaheuristicOpt R package.

Algorithm Name/Option	Description of the `algorithm` parameter
ABC	Artificial Bee Colony Algorithm
ALO	Ant Lion Optimizer
BA	Bat Algorithm
BHO	Black Hole Based Optimization Algorithm
CLONALG	Clonal Selection Algorithm
CS	Cuckoo Search Algorithm
CSO	Cat Swarm Optimization Algorithm
DA	Dragonfly Algorithm
DE	Differential Evolution Algorithm
FFA	Firefly Algorithm
GA	Genetic Algorithm
GBS	Gravitation Based Search Algorithm
GOA	Grasshopper Optimization Algorithm
GWO	Grey Wolf Optimizer
HS	Harmony Search Algorithm
KH	Krill Herd Algorithm
MFO	Moth Flame Optimizer
PSO	Particle Swarm Optimization
SCA	Sine Cosine Algorithm
SFL	Shuffled Frog Leaping Algorithm
WOA	Whale Optimization Algorithm

the corresponding values of responses and ensuing fitness. In other words, a standard indexed stream of information, like a vector or list, becomes the genetic identity of each decision. More often, this encoding is binary, but representation via real numbers is also possible in GA and mainstream in DE;

- A random sample of individuals (population) is created for initialization. Therefore, sample size in itself is a configuration parameter important to balance random diversity and computational effort;

- Individuals within the initial population, and also thereafter, are ranked according to a scaling function intended to systematically gauge their relative fitness;

- A fitness-based selection operation will sub-sample from the original population to propagate features belonging to the fittest individuals throughout the subsequent steps;

- Crossover of the selected individuals, generating new ones (subsequent generation) that combine features associated with the superior fitness driving the selection step;

- Mutation (aleatory changes into randomly chosen features of random individuals) to increase the stochastic element of the overall decision-making process;

- Check of termination criteria, which are not formal convergence metrics but rather a heuristic verification of improvement vis-Ã -vis the ability to afford additional computational effort. At this point, the cycle is either finalized or a new generation starts back into the selection step and onwards.

Algorithm 1 summarizes the outline above in a computer centered manner, with Figure 3 being its flowchart counterpart.

Algorithm 1: Genetic Algorithm.

```
Generate initial population
while termination criteria not satisfied do
    Calculate the fitness of each element in population
    for n steps do
        Use fitness to select a pair of elements for the new generation
        Create new elements e_1 and e_2
    end
    Randomly mutate some elements
    Evaluate fitness of new elements
    New population ← elements with best fitness among new elements and
     current population
end
```

Code 4 implements a genetic algorithm function call in R (Scrucca [2013]) to maximize the negative of the 2-dimensional Rastrigin function available through

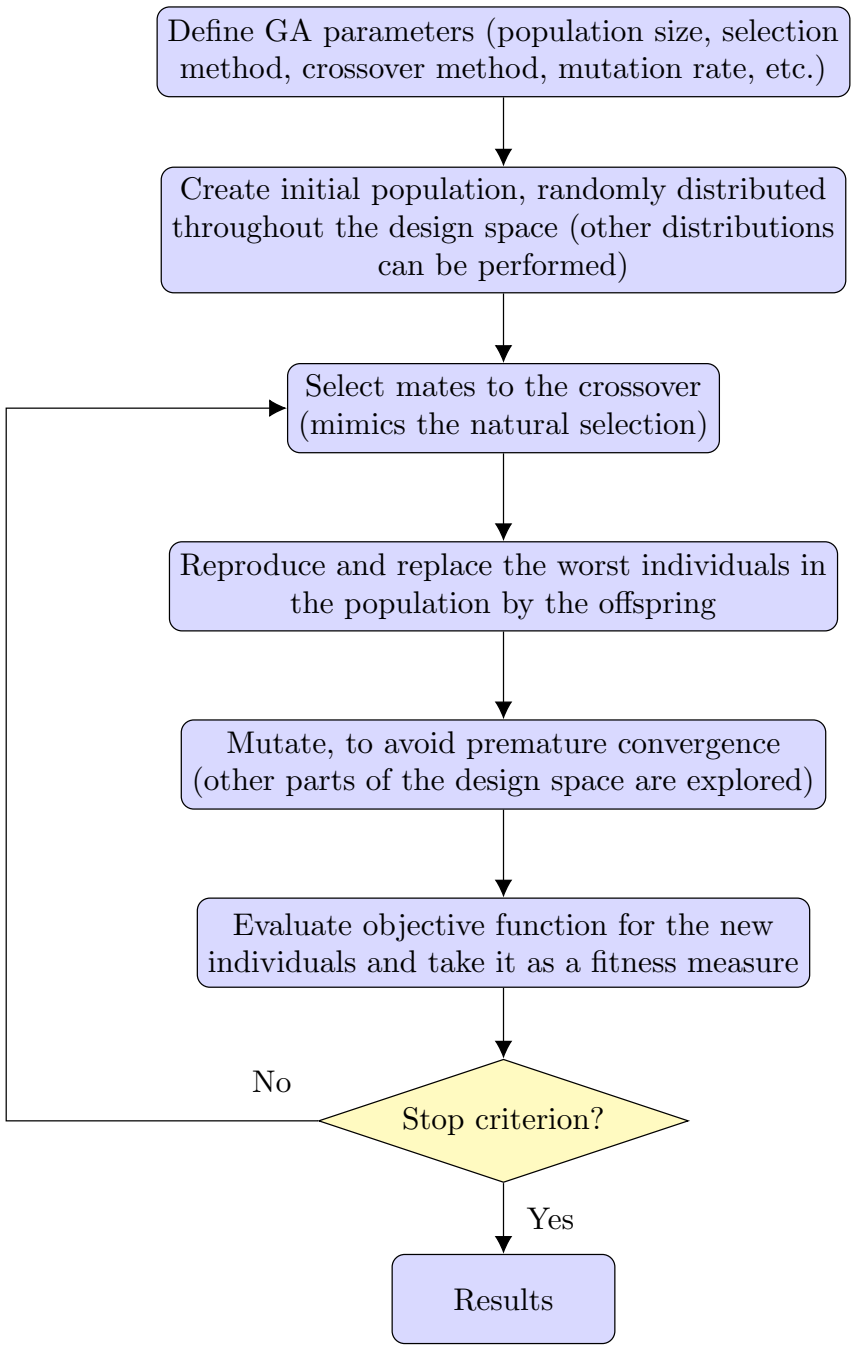


Figure 3: Genetic Algorithm flowchart.

Code 4: R implementation calling Genetic Algorithms to maximize the 2D version of the Rastrigin function as an unconstrained objective.

```
# Input #
obj2max <- function(x)
{
  obj = 20 + x[1]^2 + x[2]^2 - 10 *
  ( cos(2*pi * x[1]) + cos( 2*pi * x[2] ) )
  return (-obj)
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)

library('GA')
GA_ans <- ga(type = "real-valued", fitness = obj2max,
  lower = lower_bound, upper = upper_bound, popSize = 20,
  maxiter = 300)
cat('\nOptimal value ', GA_ans@fitnessValue,
  '\nOptimal design ', GA_ans@solution)

# Output #
Optimal value  -2.1734e-06
Optimal design  8.811057e-05 5.64903e-05
```

Code 2. The results of one function call are summarized in Figure 4, and are expected to vary from run to run due to the underlying stochastic nature of heuristic optimization methods (if affordable, a reasonable amount of repetitions should establish a stable enough trend for each application in context).

Figure 4 further illustrates the populational nature of the method, with a distribution of results at each iteration (generation) of the search. Within these distributions, some individuals are fitter than others, and statistical summaries such as the mean and the median are useful to characterize how the fitness of the entire population converges towards the intended maximum.

4.2 Evolutionary Program

Another heuristic optimization method that operates similarly to Gas, Evolutionary Program (EP) relies on the behavioral linkage between parents and their offspring, as outlined in these 3 fundamental steps (De Jong [2016]).

- Randomly initialize the initial population, whose size strongly influences the speed of optimization. The caveat is the absence of definite rules, which means that practical applications shall require some level of experimentation for fine tuning the nest tradeoff between speed and maximum coverage of loci which might contain the global optimum;
- A second generation clones the initial one, such that a spectrum of various mutation intensities is applied over the offspring solutions, according to a

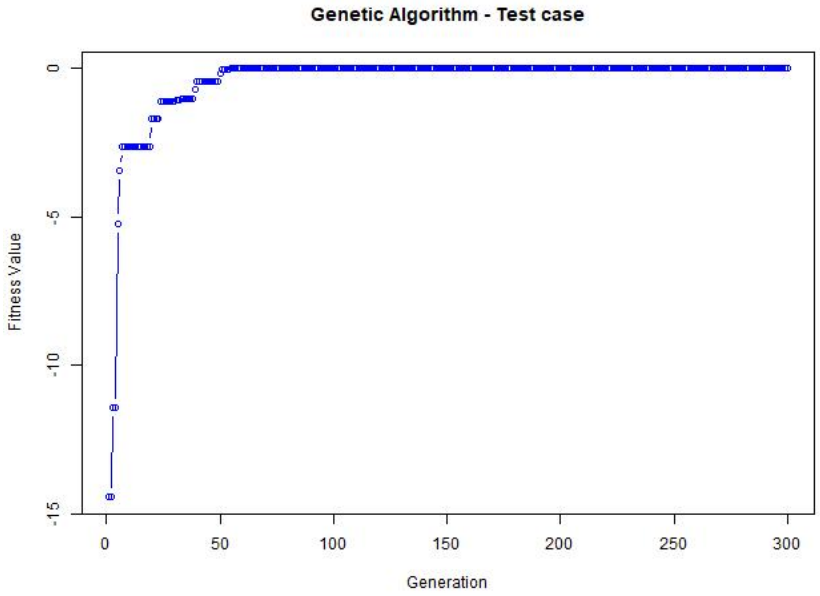


Figure 4: Graphic Output highlighting maximization of the objective along iterations.

distribution of mutation types, according to how strongly they affect the fitness of the parents (initial population);

- An arbitrarily chosen number of solutions (keeping or varying the population size determined in the initialization) within a fitness rank is for the next generation.

Algorithm 2 formalizes these steps, while Code 5 contains both the input and output to the corresponding routine in R (Bossek [2017]), in text format.

Algorithm 2: Evolutionary Programming.

Given the parameters a and b
Generate initial population
while *termination criteria not satisfied* **do**
  Calculate the fitness of each element in population
  for *each individual* **do**
    Generate a random vector $n_i \in \text{Gaussian}(0, 1)$
    New $x_i \leftarrow x_i + n_i * \sqrt{a * f(x_i) + b}$
  end
  Evaluate fitness of new elements
  New population $\leftarrow$ elements with best fitness among new elements and current population
end

A striking feature of the `ecr` library (Bossek [2017]) is the ability to easily incorporate user-defined functions for fitness evaluation, mutation strategies, and stop-

Code 5: R implementation calling Evolutionary Programming Optimization to minimize the 2D version of the Rastrigin function as an unconstrained objective.

```
# Input #
obj2min <- function(x)
{
  return( 20 + x[1]^2 + x[2]^2 - 10 *
    ( cos(2*pi*x[1]) + cos(2* pi * x[2]) ) )
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)

library('ecr')
EC_ans = ecr(fitness.fun = obj2min,
  representation = 'float',
  n.dim = 2, n.objectives = 1, survival.strategy = 'plus',
  lower = lower_bound, upper = upper_bound, mu = 20,
  lambda = 10, mutator = setup(mutGauss, sdev = 2,
  lower = lower_bound, upper = upper_bound),
  terminators = list(stopOnIters(300)))
cat('\nOptimal␣value␣', EC_ans$best.y,
  '\nOptimal␣design␣', EC_ans$best.x[[1]][1:2])

# Output #
Optimal value  1.059243
Optimal design  0.01795939 0.9936067
```

ping criteria, among others. Therefore, this library would be an adequate platform for development and evaluation of new evolutionary strategies.

4.3 Simulated Annealing

Annealing is a term from metallurgy used to describe a process in which a metal is heated to a high temperature, inducing strong perturbations to its atom?s positions. Providing that the temperature drop is slow enough, the metal will eventually stabilize into an orderly structure and, otherwise, an unstable atom structure arises.

Simulated annealing can be performed in design optimization by randomly perturbing the decision variables and keeping track of the best resulting objective value. After many tries, the most successful design is set to be the center about which a new set of perturbations will take place. In an analogy to the metallurgical annealing process, let each atomic state (design variable configurations) result in an energy level (objective function value) E. In each step of the algorithm, the atoms positions are given small random displacements due to the effect of a prescribed temperature T (standard deviation of the random number generator). As an effect, the energy level undergoes a change ΔE (variation of the objective function value). If $\Delta E \leq 0$, the objective stays the same or is minimized, thus the displacement is accepted, and the resulting configuration is adopted as the starting point of the next step. If $\Delta E > 0$, on the other hand, the probability that the new configuration is accepted is given by Equation (5):

$$P(\Delta E) = e^{\Delta E / k_b T} \tag{5}$$

where k_b is the Boltzman constant, set equal to 1. Since the probability distribution in Equation (5) is chosen, the system evolves into a Boltzman distribution. The random numbers r are obtained according to a uniform probability density function in the interval $(0, 1)$. If $r < P(\Delta E)$ the new configuration is retained. Else, the original configuration is used to start the next step.

The temperature T is simply a control parameter in the same units as the objective function. The initial value of T is related to the standard deviation of the random number generator, whilst its final value indicates the order of magnitude of the desired accuracy in the location of the optimum point. Thus, the annealing schedule starts at a high temperature which is discretely lowered (using a factor $0 < rt < 1$) until the system is "frozen", hopefully at the optimum, even if the design space is multimodal.

Similarly to previous sections, Algorithm 3 reflects the computational aspects of this procedure (Xiang et al. [2013]), actually implemented as in Code 6 to create the numeric and graphic outputs shown in panels (a) and (b) of Figure 5, respectively.

4.4 Ant Colony Optimization

As directly alluded to by its name, the natural analogy underpinning this method mirrors how ants move between their colonies and sources of food (Blum [2005]). Major components of this process are a) ants should minimize the total path travelled to gather food into their colony; b) ants may not be able to carry all the food

Algorithm 3: Simulated Annealing.

Set initial temperature $T > 0$
Cooling function: $C(T) \in [0, T]$
Generate initial population
$x_0 \leftarrow$ optimal design
while *termination criteria not satisfied* **do**
  Generate a new candidate design x_1
  if $f(x_1) > f(x_0)$ **then**
    $u \leftarrow \text{Uniform}(0, 1)$
    if $u < e^{([f(x)-f(x_1)]/T)}$ **then**
      New $x_0 \leftarrow x_1$
    end
  else
    New $x_0 \leftarrow x_1$
  end
  $T = C(T)$
end

Code 6: R implementation calling Simulated Annealing to minimize the 2D version of the Rastrigin function as an unconstrained objective.

```
# Input #
obj2min <- function(x)
{
  return( 20 + x[1]^2 + x[2]^2 - 10 *
    (cos(2*pi*x[1]) + cos(2*pi*x[2]) ) )
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)
library('GenSA')
SA_ans = GenSA(fn=obj2min, lower=lower_bound,
  upper=upper_bound, control = list(maxit = 300))
cat('\nOptimal␣value␣', SA_ans$value,
  '\nOptimal␣design␣', SA_ans$par)

# Output #
Optimal value 0
Optimal design -5.591672e-12 3.819151e-14
```

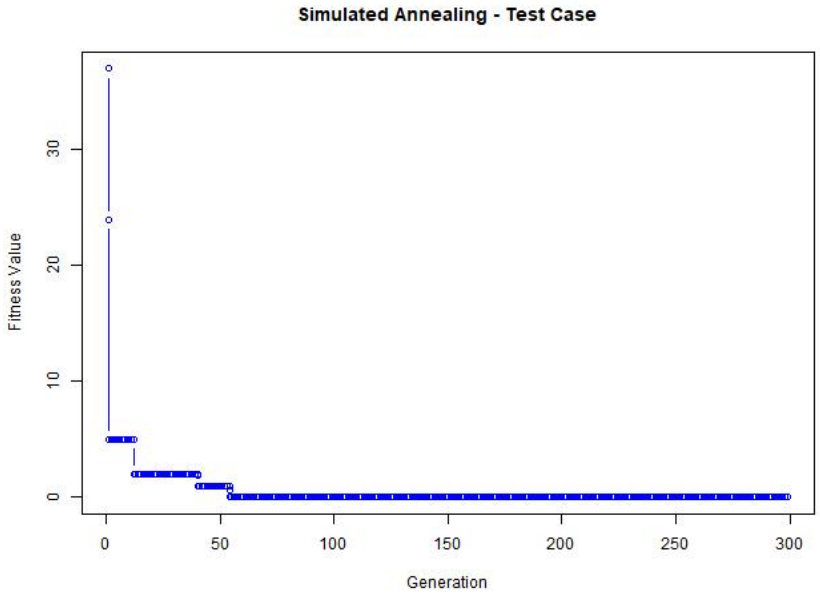


Figure 5: Graphic Output highlighting minimization of the objective along iterations.

available to the colony at once, and should need a pheromone trail (memory) to find their way back to the remaining food at the source and c) in subsequent excursions, ants will follow the shorter paths, since they result more dense on pheromones. Mathematically, this key step "c" corresponds to increasing the probability associated with a certain solution path, given its relative success as measured by earlier experimentations (when the method is exploring the decision space). In some variations, the pheromone trail is updated in each movement of an ant from one location to another, while others opt to update after all ants completed their tour. Simplifying this search into a 2-dimensional map for clarity, Equation (6) represents the amount of pheromone deposited T at a location (x, y) as a function of itself in the previous iteration $(i-1)$, discounting an evaporation rate ρ (loss of memory in the process) and adding the reinforcement of each ant (counted by k) that travels to the same location to gather food.

$$\tau_{x,y}^{(i)} = (1-\rho)\tau_{x,y}^{(i-1)} + \sum_{k} \Delta\tau_{x,y}^{k} \tag{6}$$

Furthermore, the additional pheromone deposited in Equation (6) is parameterized as follows:

$$\Delta\tau_{x,y}^{k} = \begin{cases} Q/L_k, & \text{if the ant reinforces the same path to } x, y \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

where Q is a calibration constant and L_k is the cost of ant k traveling that arc into location (x, y). The incentive/penalty is then balanced and measured along the iterations, as the commensurate amount of pheromone is deposited. Algorithm 4 describes this mechanism from the computational perspective:

Algorithm 4: Ant Colony Optimization.

```
Given the population of ants, the problem dimension and the discretized
 interval
Evaluate fitness of initial population
while termination criteria not satisfied do
    for each ant do
        for each dimension do
            for each discretized interval do
                Set probability p
            end
            Update pheromone index with probability p
        end
    end
    C_i ← cost of solution found by ant i
    for each dimension do
        for each discretized interval do
            for each ant do
                Update pheromone quantity according to pheromone index
            end
        end
    end
end
The optimal design is represented by the interval with biggest pheromone
 concentration
```

4.5 Particle Swarm Optimization

The natural analogy underpinning this method is that of the behavior and motion of collectives of animals, such as birds and fish (Mercangöz [2021]). Individual animals within the swarm are like alternative solutions known as "particles", without knowing which one is the best, but being able to measure each particle?s fitness. Per Equation (8), from each iteration i to the next $(i+1)$, the position P of each particle k is updated by a velocity term V, itself defined in Equation (9):

$$P_k^{(i+1)} = P_k^{(i)} + V_k^{(i+1)} \tag{8}$$

$$V_k^{(i+1)} = \omega \cdot V_k^{(i)} + c_1 \cdot r_1 \cdot \left(P_{\text{BEST}}^{(i)} - P_k^{(i)}\right) + c_2 \cdot r_2 \cdot \left(P_{\text{GLOBAL\_BEST}}^{(i)} - P_k^{(i)}\right) \tag{9}$$

The update of the velocity term occurs by adding up 3 terms: 1) the inertia (pondered by ω), which factors the preference for a current solution/basin of attraction; 2) the individual best (pondered by c_1 and r_1), connected to the best solution/basin of attraction ever attained by an individual particle k and 3) the collective or social best (pondered by c_2 and r_2), connected to the best solution/basin of attraction ever attained by any individual. As summarized in Equation (10), both r_1 and r_2 vary along a small range (typically the unit interval $[0, 1]$) and are unique search hyperparameters for each particle and each iteration.

Parameters c_1 and c_2, on the other hand, may take any positive real value and work as the elitism factor in Genetic Algorithms (the larger they are, the higher is the weight attributed to the best solution found up to that point of the search, which reduces the exploration into alternative basins of attraction).

$$r_{1,2} \in [0, 1],\ c_{1,2} \in \mathbb{R}^+ \tag{10}$$

The method is described in computational terms by Algorithm 5 and implemented following Bendtsen [2012] in Code 7, with both numeric and visual outputs displayed in Figure 6.

Algorithm 5: Particle Swarm Optimization.

```
Define neighborhood, maximum influence and maximum velocity
Generate initial population
Set each element velocity vector
x_0 ← optimal design
while termination criteria not satisfied do
    for each individual do
        Compute the velocity of neighbors
        Update the velocity of the element
        x_1 ← new position of the element
    end
end
x_0 = min(f(x_0), f(x_1))
```

Code 7: R implementation calling Particle Swarm Optimization to minimize the 2D version of the Rastrigin function as an unconstrained objective.

```
# Input #
obj2min <- function(x)
{
  return(20 + x[1]^2 + x[2]^2 - 10 *
    (cos(2*pi*x[1]) + cos(2* pi * x[2])))
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)
library('pso')
PSO_ans = psoptim( par = rep(NA,2), fn = obj2min,
  lower = lower_bound, upper = upper_bound,
  control = list(trace=1, maxit=300, trace.stats=TRUE))
cat('\nOptimal value ', PSO_ans$value,
  '\nOptimal design ', PSO_ans$par)

# Output #
Optimal value 0
Optimal design 6.107478e-11 -3.494791e-10
```

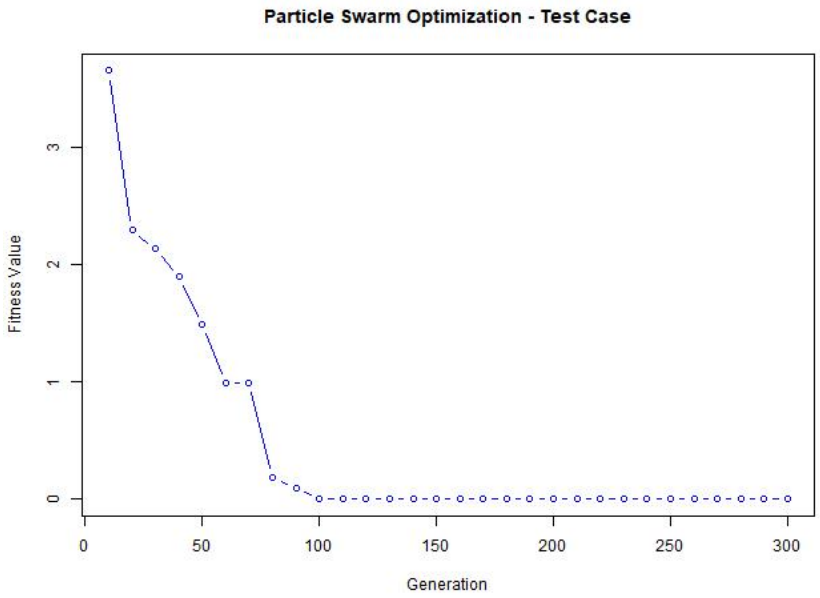


Figure 6: Graphic Output highlighting minimization of the objective along iterations.

4.6 Differential Evolution

Expanding along the lines of GA, DE is also population based stochastic technique, that is used to optimize non-linear (or even discontinuous), non-differentiable problems which are otherwise difficult to solve using classical optimization techniques. Among the GA/DE similarities, there is the reliance on the same list of core operators (selection, crossover, mutation). Also, the fitness evaluation is done for the current generation, serving as guidance for the creation of the subsequent ones. However, unlike GA, where crossover is performed initially and later followed by mutation, DE mutates the individuals prior to a recombination step which is, in essence, crossover. This particular sequence is formally captured in Algorithm 6 and may be better visualized in Figure 7. Code 8 contains its implementation (Mullen et al. [2011]) within the R statistical platform to minimize the 2-dimensional Rastrigin function, with results of a representative run summarized in Figure 8.

Algorithm 6: Differential Evolution.

```
Generate initial population
while termination criteria not satisfied do
    for each element x_i in population do
        Set random parameters
        Compute mutation of current design
        for each dimension do
            Apply mutation with a given probability
        end
        Define new design n_i
        if f(n_i) < f(x_i) then
            x_i ← n_i
        end
    end
end
```

5 Miscellaneous novel methods operating upon nature inspired analogies

Over the past decades, many different meta-heuristic algorithms have been proposed for complex optimization problems such as Genetic Algorithms (GA), Ant Colony Optimizaton (ACO) and Particle Swarm Optimization (PSO). Others new nature-inspired algorithms have emerged in recent years in order to optimize complex multimodal objective functions such as Firefly algorithm (FA), Sunflower Optimization (SFO), Bat algorithm (BA), Grey Wolf Optimization (GWO), Lichtenberg Algorithm (LA) and many others.

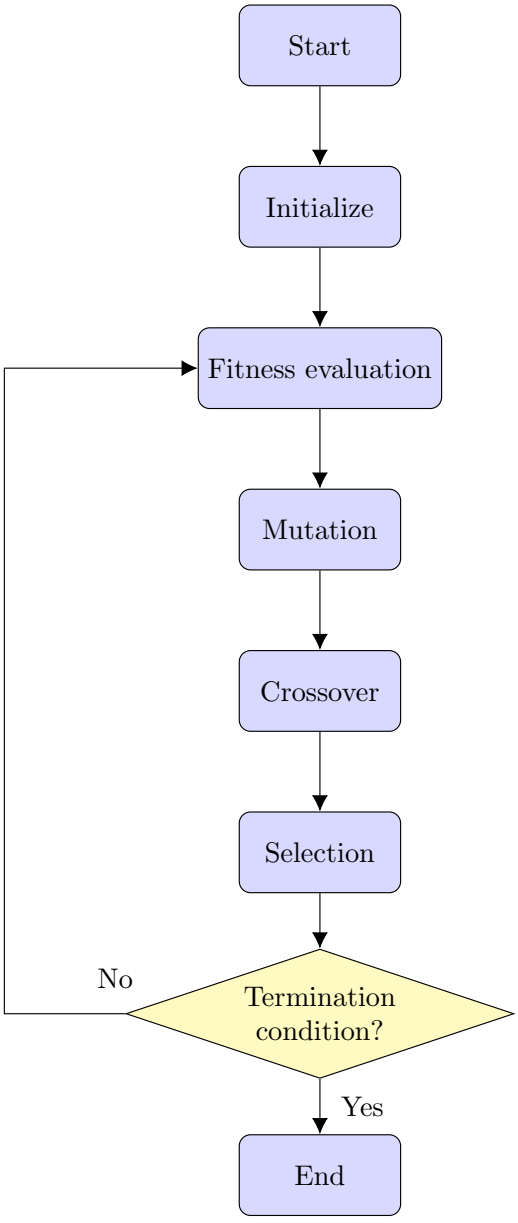


Figure 7: Schematic representation of the DE workflow.

Code 8: R implementation calling Differential Evolution to minimize the 2D version of the Rastrigin function as an unconstrained objective.

```
# Input #
obj2min <- function(x)
{
  return( 20 + x[1]^2 + x[2]^2 - 10 *
    (cos(2*pi*x[1]) + cos(2*pi*x[2])) )
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)
library('DEoptim')
DE_ans = DEoptim( fn = obj2min, lower = lower_bound,
  upper = upper_bound,
  control = list( NP = 20, itermax = 300, trace = T))
cat('\nOptimal value ', DE_ans$optim$bestval,
  '\nOptimal design ', DE_ans$optim$bestmem)

# Output #
Optimal value 1.445954e-12
Optimal design -3.099187e-08 -7.951384e-08
```

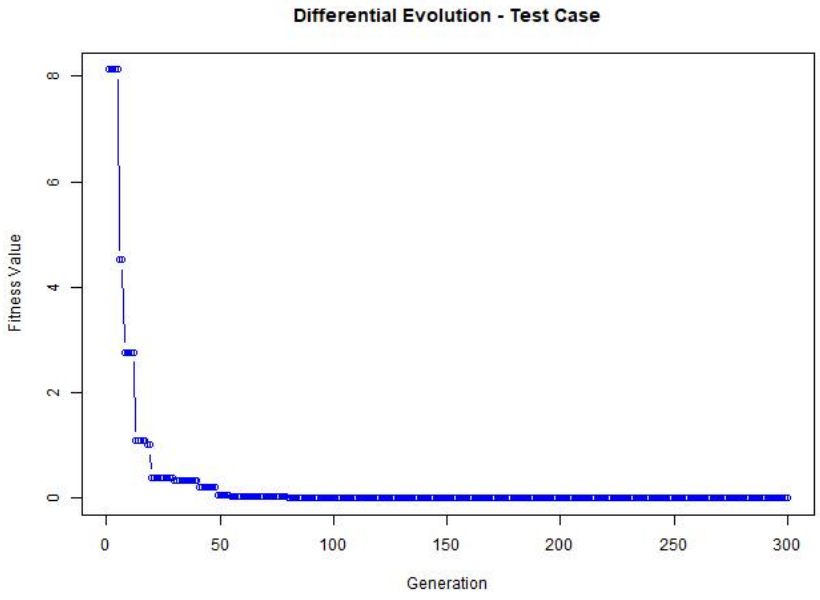


Figure 8: Graphic output highlighting minimization of the objective along iterations.

5.1 Sunflower Optimization Method

The Sunflower Optimization (SFO) method, proposed by Gomes et al. [2019], is a metaheuristic optimization method that is based on the flower pollination process proposed by Yang (2010) with the addition of a movement of plants towards the sun, which increases the convergence speed of the method, in relation to its predecessor. A population of N_{pop} plants containing parameters between a specified lower and upper limit is randomly created. The sun is assumed to be the best plant among those generated. At the beginning of each iteration $m(\%)$ of the plants will die and give way to new plants generated randomly, $p(\%)$ of the plants will pollinate each other and give rise to the new plants according to Equation 11. In addition, Algorithm 7 shows the pseudocode of the SFO methodology (Gomes and de Almeida [2020]).

$$\mathbf{x}_{i,j+1} = \mathbf{x}_{i+1,j} + rand(\mathbf{x}_{i,j} - \mathbf{x}_{i+1,j}) \tag{11}$$

The random function multiplies the vector, term-to-term by a random value between 0 and 1 generated evenly. The rest of the plants will take steps towards the sun. The direction of plants in the sun will be the second displayed in Equation 12.

$$\mathbf{s}_i = \frac{x - x_i}{||x - x_i||},\ i = 1, 2, ..., N_{pop} \tag{12}$$

The step of each plant will be calculated according to Equation 13 and the maximum step is calculated in Equation 14.

$$d_i = \lambda \cdot P_i(||x_i + x_{i-1}||) \cdot ||x_i + x_{i-1}|| \tag{13}$$

$$d_{max} = \frac{||x_{max} - x_{min}||}{2 \cdot N_{pop}} \tag{14}$$

Finally, the new plantation will be calculated by Equation 15.

$$\mathbf{x}_{i,j+1} = \mathbf{x}_{i,j} + d_i \cdot \mathbf{s}_i \tag{15}$$

The sunflower optimization source code in MATLAB language can be found in Gomes [2021].

5.2 Lichtenberg Spectrum Algorithm

The Lichtenberg algorithm (LA) was recently developed by Pereira et al. [2021]. It is inspired in the physical phenomenon of radial propagation of an intra-cloud lightning. The author used the Lichtenberg figures (LF) as a model for the construction of the optimizer. Figure 9 shows an example of LF.

A theory that is called Diffusion-Limited Aggregation (DLA) based on cluster growth can be used to describe an FL. This model is based on delimiting a region and fixing a particle in the center. A second particle is added at random and moves towards the fixed point, adhering and becoming part of the cluster. This happens with n particles until the figure is completely formed.

Algorithm 7: Sunflower Optimization Method.

A random population begins with n plants
Find the sun (Individual aiming closer to zero)
while *($k <$ Maximum number of iterations)* **do**
  $p(\%)$ of plants pollinate each other
  $m(\%)$ of the plants are removed and new random plants will be generated
  Or remaining plants will pollinate around the sun
  Evaluates new individuals
  if *(New individual is better than its predecessor)* **then**
    The new individual is stored in place of the old
  end
  if *(New individual is a great overall)* **then**
    Updates the sun
  end
end
Best solution found

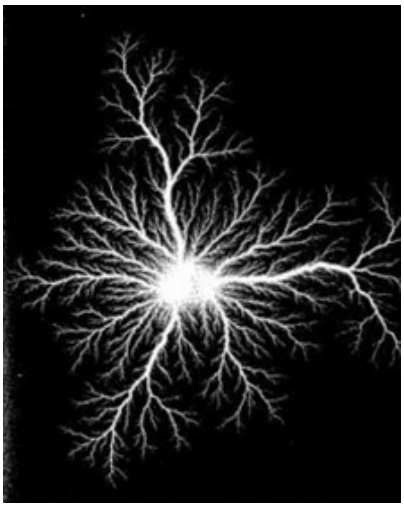
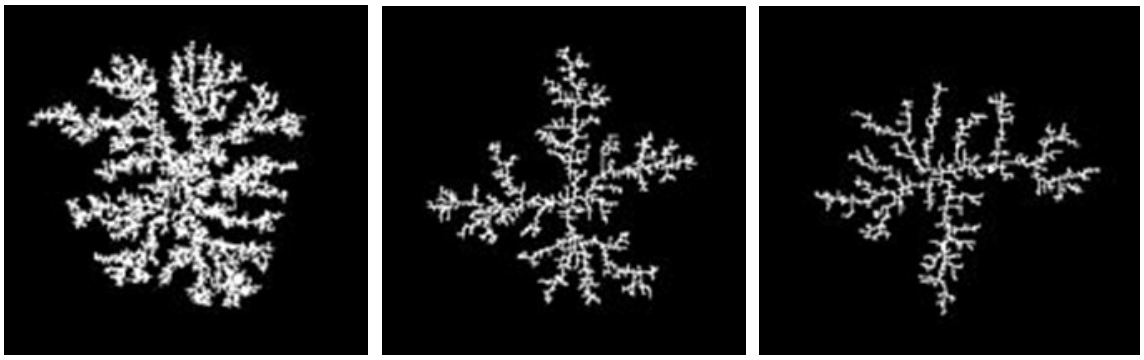


Figure 9: FL in tetra-fluoride gas under $30kV$ voltage and $30.3atm$ pressure (Pereira et al. [2021]).



(a) $S = 0.1$ (b) $S = 0.5$ (c) $S = 1$

Figure 10: Influence of the Adhesion Coefficient S on the density of the cluster.

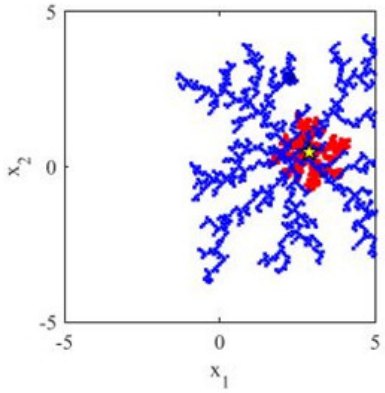


Figure 11: Local Figure (red) with 30% of the global size (blue).

Every time a particle meets the cluster, a random value is generated that is compared with an adhesion coefficient S. If the generated value is less than S, the particle is fixed, otherwise, it escapes. The lower the S, the less likely the particles will join the cluster and the greater the density. Figure 10 shows the LF formed with different values of S.

Some points about the LA must be highlighted: the maximum number of points used to build the LF is one of the input variables. If the figure exceeds the dimension of the search space, the algorithm is finalized before using all points; A random variable between zero and one is used to determine the scale factor of the figure, i.e., the size of the LF is different in each interaction, being delimited by the search space; A random factor is used to rotate the figure, avoiding repeated points; A refinement factor is used to create a second LF with the same trigger point, same rotation and smaller scale factor than the primary figure, helping to improve the local search. Figure 11 shows an example of global LF (blue) and local LF (red).

In this way, LA generates several ramifications for each iteration, always looking for the optimum point. LA has been tested in some cases with excellent results. Thus, this algorithm will be used to evaluate its performance and the already established PSO will be used too. Figure 12, adapted from Pereira et al. [2021], shows a

flowchart summarizing the functioning of the Lichtenberg algorithm. In addition, A summary of the algorithm code can be seen through the pseudocode in Algorithms 8 and 9.

In addition to Algorithm 8, the Lichtenberg Algorithm optimization source code in MATLAB language can be found in Pereira et al. [2021].

5.3 Firefly Optimization Algorithm

Based on the behavior of fireflies attracted by light sources, Yang [2009] proposed the Firefly Algorithm that has its pseudo-code shown in Algorithm 10.

In this algorithm, there are two central questions: the variation of the brightness intensity and the attractiveness of each individual. In a simple way, one can assume that the attractiveness of a firefly is determined by its brightness, which is directly related to the value of the objective function. As in the real case, it will be considered that the medium absorbs the intensity of the light emitted by the fireflies, based on the distance between each individual. It is called the light absorption coefficient. The expression for intensity I is given by Equation 16.

$$I(r) = I_0 e^{-\lambda r^2} \tag{16}$$

where I_0 is the original intensity of the light and r is the distance between the fireflies. The attractiveness β, which is proportional to the intensity of light and can be defined and written as shown in Equation 17.

$$\beta(r) = \beta_0 e^{-\lambda r^2} \tag{17}$$

where β_0 is the attractiveness at $r = 0$.

The distance between two fireflies i and j in x_i and x_j, respectively, is the Cartesian distance given by Equation 18.

$$r_{ij} = ||x_i - x_j|| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2} \tag{18}$$

where $x_{i,k}$ is the $k-th$ component of the space coordinate x_i of the $i-th$ firefly. The motion of a firefly i is determined by a more attractive (bright) firefly j and is defined by Equation 19.

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha(\text{rand} - 1/2) \tag{19}$$

where the second term represents attractiveness while the third term determines randomness, α being a parameter of randomness. The expression *rand* is a random number generator evenly distributed in $[0, 1]$. In addition, some studies indicate that the FA is particularly suited for parallel implementation and may outperform existing algorithms, such as PSO, GA, SA, and Differential Evolution, in terms of efficiency and success rates.

Leveraging implementations available at the R statistical computing platform, Code 9 allows application of the Firefly method for the minimization of the Rastrigin

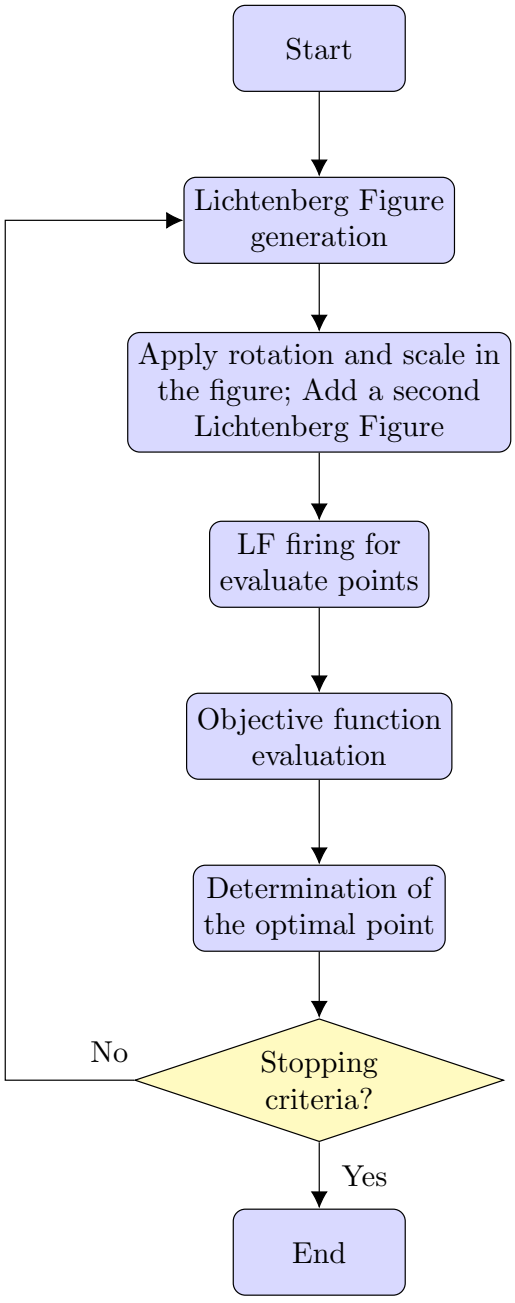


Figure 12: Lichtenberg Algorithm Flowchart.

Algorithm 8: Lichtenberg Spectrum Optimization Method - Main code.

Main:
Set objective function and search space J, upper and lower bounds
Set number of iterations and population N_{iter}, Pop (common to all optimizers)
Set Refinement and Parameter for changing the LF: Ref, M (LA routine parameters)
Set LF Parameters R_c, N_p, S
if *M = 2* **then**
  Load LF
end
if *M = 0* **then**
  Create a LF
end
while *(iter ¡ Niter)* **do**
  if *M = 1* **then**
    Create a LF
  end
  $X_{trigger} \leftarrow$ search space center (trigger point of the first LF)
  if *ref = 0* **then**
    Apply random scale and rotation
    Initialize random population through LF, X_i $(i = 1, 2, ..., Pop)$
  else
    copy LF to create a second LF of size $ref * LF(Local)$
    Apply the same random scale and rotation to both
    Initialize global random population through LF,
     $Xglobal_i$ $(i = 1, 2, ..., 0.4 * Pop)$
    Initialize local random population through LF,
     $Xlocal_j$ $(j = 1, 2, ..., 0.6 * Pop)$
    $X_i = Xglobal_i + Xlocal_j$
  end
  Calculate the fitness of each X_i
  $Xbest \leftarrow$ the lowest X_i value found
  $Xtrigger \leftarrow Xbest$
  $iter = iter + 1$
end
return $Xbest$

Algorithm 9: Lichtenberg Spectrum Optimization Method - Sub-routine creation of LF.

```
Sub-routine: creation of LF
Create an matrix of R_c sized zeros
Place a unitary particle in its center
while (i < Np) do
    Randomly place a unitary particle in the matrix
    if the plotted unitary particle t is next to another unitary particle then
        if rand < S then
            This new unitary particle is placed in the matrix
            i = i + 1
        else
            The plotted unitary particle is eliminated
        end
    end
    if the cluster of unitary particles reaches R_c then
        The simulation is finished
    end
end
X ← coordinates of all unitary particles for Cartesian space in the size of
 the search space.
```

Algorithm 10: Firefly Optimization Algorithm.

```
Objective function f(x), x = (x_1, x_2, ..., x_n)^T
Generate an initial population of n fireflies x_i, (i = 1, 2, ..., n)
Light intensity I_i at x_i is determined by f(x_i)
Define light absorption coefficient γ
while (t < MaxGenerations) do
    for i = 1 : n (all n fireflies) do
        for j = 1 : n (inner loop: all n fireflies) do
            if (I_i < I_j) then
                Move firefly i towards j
            end
            Vary attractiveness with distance r via e^{-γr^2}
            Evaluate new solutions and update light intensity
        end
    end
    Rank the fireflies and find the current global best g*
end
Postprocess results and visualization
```

Code 9: R implementation calling the Firefly optimization method to minimize the 2D version of the Rastrigin function as an unconstrained objective.

```
# Input #
obj2min <- function(x)
{
  return( 20 + x[1]^2 + x[2]^2 - 10 *
    (cos(2*pi*x[1]) + cos(2*pi*x[2])) )
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)
library('metaheuristicOpt')
FFA_ans = metaOpt(obj2min, optimType = 'MIN',
  algorithm = 'FFA', numVar = 2,
  rangeVar = rbind(lower_bound, upper_bound),
  control = list(numPopulation=20, maxIter=700))
cat('\nOptimal␣value␣', FFA_ans$optimumValue,
  '\nOptimal␣design␣', FFA_ans$result)

# Output #
Optimal value 4.510703
Optimal design 0.9207517 1.081387
```

Function, in the present case presenting both input and output (a dedicated post-processing module is still to be implemented to enable the generation of graphical output, such as the value of the objective function over iterations, for example).

5.4 Bat Optimization Algorithm

The Bat Optimization Algorithm (BA) is inspired in the micro-bats behavior and its ability of echolocation. This algorithm uses as parameters the loudness and pulse emission, it was also the first to take and account the frequency tuning.

Bats utilize a type of sonar, called echolocation, to recognize prey, stay away from obstacles, and locate their roosting crevices in the obscurity. These bats emanate a loud sound pulse tune in for the reverberation that bounces back from the surrounding objects. Their pulses fluctuate in properties which can be corresponded with their chasing methodologies and strategies, depending on the species. Most bats utilize short, frequency-modulated signals to sweep through about an octave; others all the more frequently utilize consistent recurrence signals for echolocation. Their sign data transfer capacity changes with species and frequently increments by utilizing more harmonics (Yang [2010]).

The BA uses the diversity of pulse emission rate r and loudness A to control exploration and exploitation characteristics. In the main BA, Equations 20, 21 and 22 present the main equations for the evolutionary process (Yang [2010]).

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{20}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x^*)f_i \tag{21}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{22}$$

where $\beta in[0,1]$ is a random vector drawn from a uniform distribution so that the frequency can vary from f_{min} to f_{max}. Equally important, these updating equations are also associated with the pulse r and loudness A via a uniformly distributed random number ϵ. Selection is done by the current best solution x^* found so far by all the bats.

It can be seen that both above equations are linear in terms of x_i and v_i. However, the control of exploration and exploitation of the BA is defined by the changes of loudness A from a high value to a lower value and the emission rate r from a lower to a higher value (Equation 23).

$$A_i^{t+1} = \mu A_i^t,\ r_i^{t+1} = r_i^0(1 - e^{-\gamma t}) \tag{23}$$

where $0 < \mu < 1$ and $\gamma > 0$ are two random parameters. As a result, the actual algorithm can have a weak nonlinearity. Consequently, BA can have a faster convergence rate in comparison with PSO. The basic steps of BA can be summarized as the schematic pseudo code shown Algorithm 11.

Algorithm 11: Bat Optimization Algorithm.

```
Initialize the bat population x_i and v_i
Initialize frequencies f_i, pulse rates r_i and the loudness A_i
while termination criteria not satisfied do
    Generate new solution by adjusting frequency
    Update velocities and locations/solutions
    if rand > r_i then
        Select a new solution
        Generate local solution around the selected
    end
    Generate a new solution by flying randomly
    if (rand < A_i) and (f(x_i) < f(x*)) then
        Accept the new solutions
        Increase r_i and reduce A_i
    end
    Rank the bats and find the current best x*
end
```

Similarly, to the previous section, the existence of an implementations available at the R statistical computing platform allows the construction of Code 10, which uses the Bat Optimization method for the minimization of the Rastrigin Function. (Once again both input and numeric output are presented, with the availability of visual results pending a dedicated post-processing module).

Code 10: R implementation calling the Bat optimization method to minimize the 2D version of the Rastrigin function as an unconstrained objective.

```
# Input #
obj2min <- function(x)
{
  return( 20 + x[1]^2 + x[2]^2 - 10 *
    (cos(2*pi*x[1]) + cos(2*pi*x[2]) ) )
}
lower_bound = c(-7, -7); upper_bound = c(7, 7)
library('metaheuristicOpt')
BA_ans = metaOpt(obj2min, optimType = 'MIN',
  algorithm = 'BA', numVar = 2,
  rangeVar = rbind(lower_bound, upper_bound),
  control = list(numPopulation = 20, maxIter = 700))
cat('\nOptimal␣value␣', BA_ans$optimumValue,
  '\nOptimal␣design␣', BA_ans$result)

# Output #
Optimal value 0
Optimal design 3.238586e-292 1.289825e-292
```

6 Practical considerations for applied optimization in general and structural optimization in particular

Noteworthy of population-based methods is the increase in computational effort resulting from simultaneous consideration of multiple designs per iteration instead of a singular one, as in the case of classical (derivative based) optimization methods.

For this reason, it is often impractical to couple heuristic optimizers directly into high-fidelity analysis codes (Finite Element and others), and it is standard practice to evaluate the functions pertaining to the optimization problem by way of approximations, such as the statistical surrogates (or metamodels) described in another Chapter of the present volume. While a faster to calculate surrogate will enable heuristic optimization methods to work properly and improve the decision making process even with the complexity added by multiple basins of attraction, attention is required to manage the propagation of approximation errors, which takes place a) preemptively, by building accurate enough approximations and b) correctively, by updating/enhancing initial versions of surrogates as needed and, in certain cases, making calls to the actual analysis software at a lower frequency (not at every iteration or step of the search process) so that error propagation can be contained. The remaining challenge is to ensure that the search is not prone to invoke a design candidate whose analysis and/or derivative calculation are not

feasible, and addressing it is often dependent upon sound judgment about physical regimes and their changes.

On a different but related note, it is not unusual that a singular heuristic search method is not the best alternative to solve a given optimization problem, regardless of how powerful (and intrinsically computationally sophisticated/expensive it might be). Hence, practitioners have developed the so-called lifecycle approaches (Viana et al. [2007]), so that combinations of optimization algorithms (calculus based and/or heuristic) are employed in tandem, subject to several configuration and transition criteria that evolve with the progress of the search itself and rely on more than one method over the duration of the optimization procedure (also known as "lifecycle", hence the approach name).

References

C. Bendtsen. pso: Particle swarm optimization. *R package version 1.0.3*, 2012. URL `https://CRAN.R-project.org/package=pso`.

C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4):353–373, 2005.

J. Bossek. Ecr 2.0: A modular framework for evolutionary computation in r. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1187–1193, 2017.

K. De Jong. Evolutionary computation: a unified approach. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 185–199, 2016.

G. F. Gomes. Sunflower optimization (sfo) algorithm. Technical report, MATLAB Central File Exchange, 2021. URL `https://www.mathworks.com/matlabcentral/fileexchange/69076-sunflower-optimization-sfo-algorithm`.

G. F. Gomes and F. A. de Almeida. Tuning metaheuristic algorithms using mixture design: Application of sunflower optimization for structural damage identification. *Advances in Engineering Software*, 2020. doi: https://doi.org/10.1016/j.advengsoft.2020.102877.

G. F. Gomes, S. S. da Cunha, and A. C. Ancelotti. A sunflower optimization (sfo) algorithm applied to damage identification on laminated composite plates. *Engineering with Computers*, 35(2):619–626, 2019. doi: https://doi.org/10.1007/s00366-018-0620-8.

B. A. Mercangöz. *Applying Particle Swarm Optimization*. Springer, 2021.

K. Mullen, D. Ardia, D. L. Gil, D. Windover, and J. Cline. Deoptim: An r package for global optimization by differential evolution. *Journal of Statistical Software*, 40(6):1–26, 2011.

J. L. J. Pereira, M. B. Francisco, C. A. Diniz, G. A. Oliver, S. S. Cunha Jr, and G. F. Gomes. Lichtenberg algorithm: A novel hybrid physics-based meta-heuristic for global optimization. *Expert Systems with Applications*, 170, 2021. doi: https://doi.org/10.1016/j.eswa.2020.114522.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL `https://www.R-project.org/`.

L. S. Riza et al. metaheuristicopt: Metaheuristic for optimization. *R package version 2.0.0*, 2019. URL `https://CRAN.R-project.org/package=metaheuristicOpt`.

L. Scrucca. Ga: a package for genetic algorithms in r. *Journal of Statistical Software*, 53(4), 2013.

F. A. C. Viana, V. Steffen Jr, S. Butkewitsch, and M. de Freitas Leal. About the optimum design of an aircraft pressure bulkhead by using multi-fidelity and lifecycle algorithm. In *IPDO-Inverse Problems, Design and Optimization Symposium*, 2007.

Y. Xiang, S. Gubian, B. Suomela, and J. Hoeng. Generalized simulated annealing for global optimization: the gensa package. *R Journal*, 5(1):13, 2013.

X.-S. Yang. Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169–178. Springer, 2009.

X.-S. Yang. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer, 2010.

X.-S. Yang. Nature-inspired optimization algorithms: Challenges and open problems. *Journal of Computational Science*, 46, 2020.