SoftwareX 18 (2022) 101083

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

EmiR: Evolutionary minimization for R

Davide Pagano *, Lorenzo Sostero

Department of Mechanical and Industrial Engineering, University of Brescia, Brescia, Italy

ARTICLE INFO

Article history:
Received 21 January 2022
Received in revised form 22 March 2022
Accepted 5 April 2022

Keywords:
Evolutionary algorithms
Optimization
R

ABSTRACT

Classical minimization methods, like the steepest descent or quasi-Newton techniques, have been proved to struggle in dealing with optimization problems with a high-dimensional search space or subject to complex nonlinear constraints. In the last decade, the interest on metaheuristic nature-inspired algorithms has been growing steadily, due to their flexibility and effectiveness. In this paper we present EmiR, a package for R which implements several metaheuristic algorithms for optimization problems. Unlike other available tools, EmiR can be used not only for unconstrained problems, but also for problems subjected to inequality constraints and for integer or mixed-integer problems. Main features of EmiR, its usage and the comparison with other available tools are presented.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Code metadata

Current code version	v1.0.3
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00026
Legal Code License	GNU General Public License (GPL)
Code versioning system used	git
Software code languages, tools, and services used	C++, R
Support email for questions	davide.pagano@unibs.it

1. Introduction

Classical optimization techniques, as the steepest descent algorithm and quasi-Newton techniques, are based on the use of differential calculus in locating the optimum solution. Although still widely used in several research areas, these algorithms have been proved to struggle in dealing with optimization problems with a high-dimensional search space or subject to complex nonlinear constraints [1]. In the last decade, the interest on metaheuristic algorithms – also referred to as evolutionary algorithms [2] – has been growing steadily. The term metaheuristic, proposed by Glover [3] in 1986, is composed by the words *heuristic*, which used to denote algorithms with stochastic components in the past, and *meta*, which means "beyond" or "higher level". Metaheuristic algorithms are therefore higher-level heuristic algorithms, in the sense they are more general in problem-solving [4] and, nowadays, are typically based on a metaphor of natural or man-made processes, like the search for food or haunting of nearly any species of animals [5]. Most of these methods evolve an initial population of individuals, each of them representing a candidate solution to the problem, toward the global minimum or maximum of the cost/fitness function, according to nature-inspired processes.

2. Software description

EmiR is a package for R, written in C++ for speed, which implements some of the most popular population-based metaheuristic algorithms: Artificial Bee Colony algorithm (ABC) [6,7], Bat algorithm (BAT) [8], Cuckoo Search (CS) [9], Genetic Algorithms (GA) [10,11], Gravitational Search Algorithm (GSA) [12], Grey Wolf Optimization (GWO) [13,14], Harmony Search (HS) [15,16], Improved Harmony Search (IHS) [17], Moth-flame Optimization (MFO) [18], Particle Swarm optimization (PS) [19,20], Simulated Annealing (SA) [21–23], Whale Optimization Algorithm (WOA) [24].

A detailed description of these algorithms is beyond the scope of this work. However, it is important to highlight that for algorithms where multiple approaches have been proposed in literature, such as SA, PS and GA, we opted for most recent ones, as described in the following. For SA we implemented the population-based version of the algorithm proposed in 2016 by Askarzadeh et al. [23]. For PS we opted for the general approach with adaptive parameters, inertia on particles and constraints on their maximum velocity [20]. Finally, for GA we decided to implement the version described by Haupt [25], characterized by high efficiency mechanisms of selection and mating.

* Corresponding author.
E-mail addresses: davide.pagano@unibs.it (Davide Pagano), l.sostero@studenti.unibs.it (Lorenzo Sostero).

https://doi.org/10.1016/j.softx.2022.101083
2352-7110/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

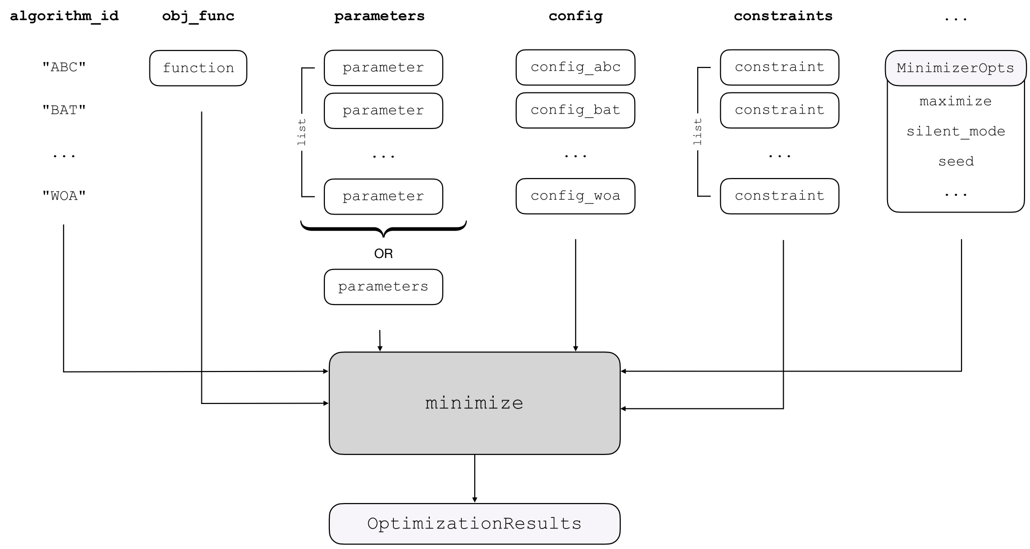

Fig. 1. Graphical representation of the function `minimize`.

The package provides a single interface to all the available algorithms, by means of the function:

```
minimize(algorithm_id, obj_func, parameters,
    config, constraints = NULL, ...)
```

whose block diagram is represented in Fig. 1. `minimize` accepts the following arguments:

- `algorithm_id` — the identification code of the algorithm to be used;
- `obj_func` — the objective function to be minimized/maximized;
- `parameters` — the list of parameters the objective function is minimized/maximized with respect to;
- `config` — the configuration parameters of the algorithm;
- `constraints` — the (optional) list of constraints the objective function is subjected to;
- `...` — additional (optional) parameters.

The choice of the algorithm to be used is performed by passing its corresponding ID (an object of class `character`) to the argument `algorithm_id`. A `data.frame` object with the list of available algorithms in EmiR, including their IDs and configuration functions, can be obtained with the function `list_of_algorithms`.

The objective function to be minimized or maximized has to be specified by means of the argument `obj_func`. Valid functions have a single vector argument for all the independent variables, as in the following example.

```
sphere_function <- function(x) {
 x1 <- x[1]
 x2 <- x[2]
 x3 <- x[3]
 value <- x1^2 + x2^2 + x3^2
 return(value)
}
```

Depending on the dimension of the objective function, one or more objects of class `Parameter` have to be defined, one for each independent variable the function depends on. This S4 class is necessary to store range, name and type (integer of continue) for each variable. The class `Parameter` is not exposed to the user, who instead uses the function `parameter`, which returns an instance of it. Arguments of function `parameter` are reported below.

```
parameter(name, min_val, max_val, integer = FALSE)
```

With `name` the user specifies the name of the parameter, with `min_val` and `max_val` its range, and with `integer` whether the variable is integer or continue. As an objective function can depend on multiple variables, a `list` of `Parameter` objects has to passed to the argument `parameters` of the `minimize` function. When dealing with high-dimensional functions, it could be tedious to create a new `Parameter` object for each variable one by one. For this reason, EmiR also implements the function `parameters`, which accepts a matrix as input and returns a list of objects of `Parameter`.

All algorithms share some common configuration parameters, in addition to those which are algorithm-specific. For this reason, a S4 class for each algorithm has been created to deal with all these different tuning variables. Those classes are not exposed to the user, who instead creates instances of them, by using the configuration functions associated to each algorithm, whose names are reported in the column `Configuration function` of the `list_of_algorithms` function's output. Only arguments `iterations` and `population_size` do not have default values and have to be specified.

EmiR can also be used for constrained optimization problems, as long as the objective function is subjected to inequality constraints. The S4 class `Constraint` has been designed to deal with constraints, but it is not exposed to the user, who instead can define a constraint by using the function `constraint(func, inequality)`, where arguments `func` and `inequality` expect respectively an object of class `function` and one of class `character` with the inequality type (">", ">=", "<=", "<"). The function passed to `func` has to represent the first term of an inequality with zero. Valid functions depend on one or more of the independent variables of the objective function, and have a single vector argument, like in the example below.

```
g1 <- function(x) {
  value <- x[1] + 2*x[2]
  return(value)
}
c1 <- constraint(g1, "<=")
```

There are also additional parameters and options that can be specified in the function minimize. Just to name a few, it is possible to choose how to handle out-of-boundary solutions (oob_solutions), and to select the approach to use in the constrained optimization (constrained_method).

Other functions in EmiR, mainly related to the graphical presentation of results, will be directly introduced in the following examples.

3. Examples of unconstrained optimization problems

In this section, few examples on how to use the function minimize for unconstrained optimization problems are presented.

3.1. Example 1: 4-dimensional miele cantrell function

In this first example, the 4-dimensional Miele Cantrell function [26] is defined and evaluated in $x_i \in [-2, 2]$, for $i = 1, \ldots, 4$:

$$f(x) = (e^{-x_1} - x_2)^4 + 100(x_2 - x_3)^6 + (\tan(x_3 - x_4))^4 + x_1^8.$$

The function has a global minimum: $f(0, 1, 1, 1) = 0$. The following code shows the minimization of this function with the Bat algorithm, using the default values for its algorithm-specific parameters. The Miele Cantrell function is one of those functions which are already the pre-defined in EmiR, whose list is accessible via the command list_of_functions().

```
p1 <- parameter("x1", -2, 2, FALSE)
p2 <- parameter("x2", -2, 2, FALSE)
p3 <- parameter("x3", -2, 2, FALSE)
p4 <- parameter("x4", -2, 2, FALSE)

conf_algo <- config_bat(iterations = 200,
                        population_size = 100)
results <- minimize(algorithm_id = "BAT",
                    obj_func = miele_cantrell,
                    parameters = list(p1, p2, p3, p4),
                    config = conf_algo,
                    seed = 1)
print(results)
```

When running the previous code, an output similar to the following one is shown.

```
     EmiR Minimization Results
---------------------------------------
      minimizer | BAT
     iterations | 200
 population size | 100
   minimum value | 7.349613e-16
 best parameters | x1 = -0.01274905
                 | x2 = 1.012874
                 | x3 = 1.013704
                 | x4 = 1.013734
---------------------------------------
```

It is possible to plot the best function value (best cost) as a function of the iteration using the following function.

```
plot_history(results, log = "y")
```

The resulting plot is shown in Fig. 2 (left). Being based on base plot function, plot_history accepts the same arguments for the customization of the plot. Best cost at each iteration can be also directly accessed by means of the slot cost_history, as in the example below, which produces the plot in Fig. 2 (right).

```
plot(151:200, tail(results@cost_history, 50),
  type = "l", xlab = "Iteration",
  ylab = "Best function value", lwd = 2)
```

3.2. Example 2: 2-dimensional ackley function

In this second example, the 2-dimensional Ackley function [27] is chosen to show some of the visualization capabilities of EmiR. The following code shows the minimization of the function, in the range $x_i \in [-30, 30]$, for $i = 1, 2$, using the Particle Swarm algorithm.

```
pars <- parameters(matrix(rep(c(-30, 30), 2), 2, 2))

conf_algo <- config_ps(iterations = 200,
                       population_size = 20)
results <- minimize(algorithm_id = "PS",
                    obj_func = ackley_func,
                    parameters = pars,
                    config = conf_algo,
                    save_pop_history = TRUE,
                    seed = 1)
```

The option save_pop_history has been set to TRUE, meaning that the position of all particles at each iteration is saved, and, because of this, it is possible to visualize how it evolves with the iteration by means of the following code, which produces the plots in Fig. 3.

```
plot_population(results, iteration = 1,
                n_points = 200)
plot_population(results, iteration = 30,
                n_points = 200)
```

Function plot_population can be used only for 1D, 2D and 3D optimization problems. Examples for 1D and 3D cases are shown in Fig. 4. Function animate_population allows the creation of animations of how the population positions evolved with the iteration.

3.3. Example 3: Unconstrained nonlinear integer problem

As already mentioned, EmiR can be also used for integer and mixed-integer optimization problems. The following objective function [28] is maximized in the range $x_i \in [0, 99]$, for $i = 1, 2, \ldots, 10$, with $x_i \in \mathbb{N}_0$.

$$f(x) = x_1^2 + x_1x_2 - x_2^2 + x_3x_1 - x_3^2 + 8x_4^2 - 17x_5^2 + 6x_6^3 + \\ + x_6x_5x_4x_7 + x_8^3 + x_9^4 - x_{10}^5 - x_{10}x_5 + 18x_3x_7x_6.$$

The problem has the following global maximum: $f(\mathbf{x}^*) = 216300719$, with $\mathbf{x}^* = (99, 49, 99, 99, 99, 99, 99, 99, 99, 0)$. The following code shows the maximization of the previous objective function, using the Whale Optimization algorithm.

```
ob <- function(x) {
  x[1]^2 + x[1]*x[2] - x[2]^2 + x[3]*x[1] - x[3]^2
  + 8*x[4]^2 - 17*x[5]^2 + 6*x[6]^3
  + x[6]*x[5]*x[4]*x[7]
  + x[8]^3 + x[9]^4 - x[10]^5 - x[10]*x[5]
  + 18*x[3]*x[7]*x[6]
}
p <- parameters(matrix(rep(c(0, 99, TRUE), 10),
  nrow=3))
conf <- config_algo(algorithm_id = "WOA",
                    population_size = 100,
                    iterations = 500)
results <- minimize(algorithm_id = "WOA",
                    obj_func = ob,
                    config = conf,
                    maximize = TRUE,
```

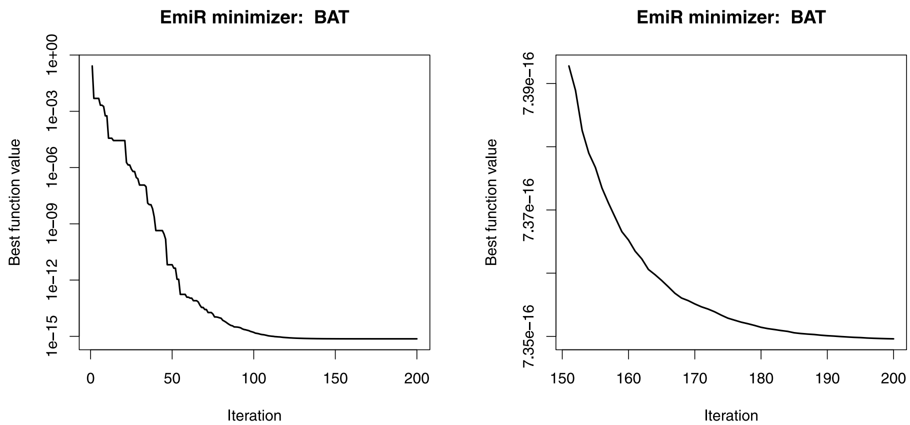

Fig. 2. Best cost as a function of all iterations, in a minimization of the 4D Miele Cantrell function (left), and of iterations in the range [151, 200] (right).

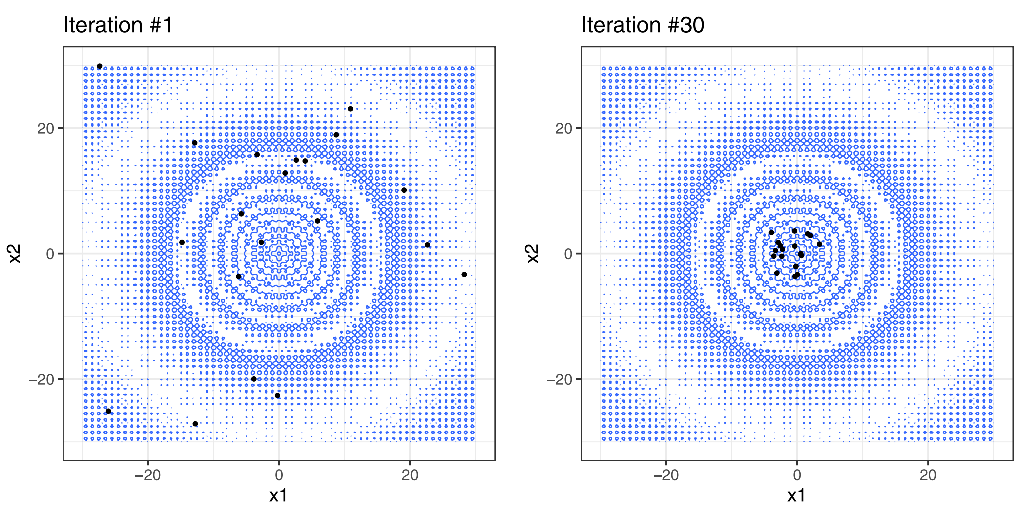

Fig. 3. Population of particles at iteration 1 (left) and iteration 30 (right), superimposed to the profile of the Ackley function, for a minimization with the Particle Swarm algorithm.

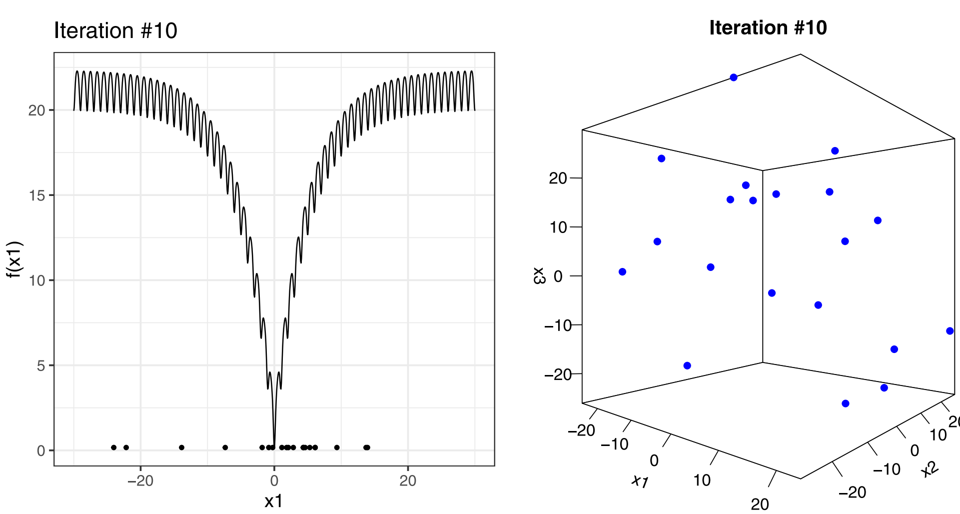

Fig. 4. Population of particles at iteration 10 for a minimization of the 1D (left) and 3D (right) Ackley function, using the Particle Swarm algorithm.

```
                  parameters = p,
                  save_pop_history = TRUE,
                  oob_solutions = "RBC",
                  seed = 10)
print(results)
```

When running the previous code, an output similar to the following one is shown.

```
    EmiR Minimization Results
---------------------------------------
      minimizer | WOA
     iterations | 500
population size | 100
  maximum value | 216300719
best parameters | x1 = 99
                | x2 = 49
                | x3 = 99
                | x4 = 99
                | x5 = 99
                | x6 = 99
                | x7 = 99
                | x8 = 99
                | x9 = 99
                | x10 = 0
---------------------------------------
```

4. Examples of constrained optimization problems

Because of their strong heuristic component, the use of evolutionary algorithms for constrained problems was very limited in the past [29]. However, for both linearly and non-linearly constrained problems with inequality constraints, evolutionary algorithms have proved to be a promising option. EmiR offers three approaches to deal with constrained problems:

- *penalty method* — the constrained problem is converted to an unconstrained one, by adding a term, called *penalty function*, to the objective function. The penalty function consists of a *penalty parameter* multiplied by a measure of the violation of the constraints. The penalty parameter is increased at each iteration;
- *barrier method* — the value of the objective function is set equal to an arbitrary large positive (or negative in case of maximization) number if any of the constraints is violated;
- *acceptance-rejection method* — all solutions at each generation are checked for possible violation of the constraints: if so, they are replaced by new randomly generated solutions in the feasible region (FR).

For all previous methods, the user can choose to generate the initial population either in the feasible region of the problem, or in the full range of its parameters.

In this section, two examples on how to use the function `minimize` for constrained optimization problems are presented.

4.1. Example C1: nonlinear programming

Let us consider the following example from [30]:

$$\begin{aligned} &\min f(x,y) = (x-10)^3 + (y-20)^3 \\ \text{subject to: } & g_1(x,y) = -(x-5)^2 - (y-5)^2 + 100 \le 0 \ , \\ & g_2(x,y) = (x-6)^2 + (y-5)^2 - 82.81 \le 0 \end{aligned} \tag{1}$$

in the interval $x \in [13, 100]$ and $y \in [0, 100]$. The coloured contour plot of the objective function, as well as the curves $g_1(x, y) = 0$ and $g_2(x, y) = 0$, and the FR are shown in Fig. 5 (left). The problem has the following global minimum: $f(14.09500, 0.84296) = -6961.81474448783$, which is located within a very narrow region of the FR, as shown in Fig. 5 (right).

The following code shows how to set up this constrained optimization problem in EmiR, using the Particle Swarm algorithm.

```
ob <- function(x) (x[1]-10)^3 + (x[2]-20)^3
g1 <- function(x) -(x[1]-5)^2 - (x[2]-5)^2 + 100
g2 <- function(x)  (x[1]-6)^2 + (x[2]-5)^2 - 82.81
c1 <- constraint(g1, "<=")
c2 <- constraint(g2, "<=")
p1 <- parameter("x1", 13, 100)
p2 <- parameter("x2", 0, 100)
conf <- config_algo(algorithm_id = "PS",
                    population_size = 200,
                    iterations = 10000)
results <- minimize(algorithm_id = "PS",
                    obj_func = ob,
                    config = conf,
                    parameters = list(p1,p2),
                    constraints = list(c1,c2),
                    save_pop_history = TRUE,
                    constrained_method = "PENALTY",
                    constr_init_pop = FALSE,
                    oob_solutions = "RBC",
                    penalty_scale = 5,
                    seed = 1)
```

When running the previous code, an output similar to the following one is shown.

```
    EmiR Minimization Results
---------------------------------------
      minimizer | PS
     iterations | 10000
population size | 200
  minimum value | -6961.80150
best parameters | x1 = 14.09500
                | x2 = 0.84297
---------------------------------------
```

4.2. Example C2: nonlinear programming with mixed-integer variables

Let us consider the following problem, based on the pressure vessel design optimization problem in [31]:

$$\begin{aligned} \min\ & f(x_1,x_2,x_3,x_4) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 \\ & \quad + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \\ \text{subject to: } & g_1(x_1,x_3) = -x_1 + 0.0193x_3 \le 0 \\ & g_2(x_2,x_3) = -x_2 + 0.00954x_3 \le 0 \\ & g_3(x_3,x_4) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \le 0 \end{aligned} \ , \tag{2}$$

in the interval $x_1 \in [1.125, 2]$, $x_2 \in [0.625, 2]$, $x_3 \in [10, 240]$ and $x_4 \in [10, 240]$, and with $x_1 \cdot x_2 = 0.0625$. The best solution reported in [31] is $f(\mathbf{x}^*) = 7199.412$, with $\mathbf{x}^* = [1.125, 0.625, 58.2895, 43.6964]$. The following code shows the implementation in EmiR of this optimization problem, using the Bat algorithm.

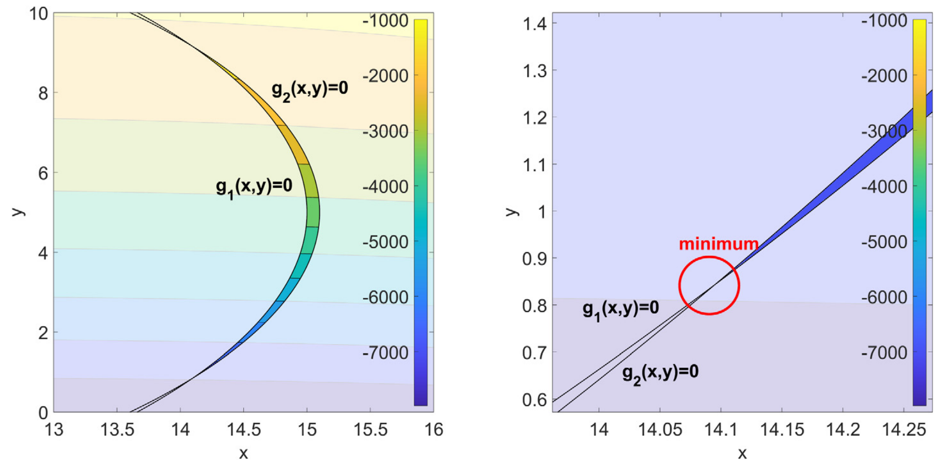

Fig. 5. Coloured contour plot of the objective function in a region around the feasible region (left) and in a region around the minimum (right). The area outside the feasible region has been desaturated. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

```
ob <- function(x) {
 0.6224*(x[1]*0.0625)*x[3]*x[4]
 + 1.7781*(x[2]*0.0625)*x[3]^2
 + 3.1611*(x[1]*0.0625)^2*x[4]
 + 19.8621*(x[1]*0.0625)^2*x[3]
}

g1 <- function(x) 0.0193*x[3] - (x[1]*0.0625)
g2 <- function(x) 0.00954*x[3] - (x[2]*0.0625)
g3 <- function(x) {
  1296000 - pi * x[3]^2 * x[4] - 4/3 * pi * x[3]^3
}
c1 <- constraint(g1, "<=")
c2 <- constraint(g2, "<=")
c3 <- constraint(g3, "<=")

p1 <- parameter("x1", 18, 32, integer = TRUE)
p2 <- parameter("x2", 10, 32, integer = TRUE)
p3 <- parameter("x3", 10, 240)
p4 <- parameter("x4", 10, 240)

conf <- config_algo(algorithm_id = "BAT",
             population_size = 500,
             iterations = 7000)
results <- minimize(algorithm_id = "BAT",
            obj_func = ob,
            config = conf,
            parameters = list(p1,p2, p3, p4),
            constraints = list(c1,c2,c3),
            save_pop_history = TRUE,
            constrained_method = "BARRIER",
            constr_init_pop = TRUE,
            oob_solutions = "RBC",
            seed = 1)
```

The corresponding output is shown below (note that $18 \cdot 0.0625 = 1.125 = x_1^*$ and $10 \cdot 0.0625 = 0.625 = x_2^*$).

```
       EmiR Minimization Results
--------------------------------------
        minimizer | BAT
       iterations | 7000
  population size | 500
    minimum value | 7199.37
  best parameters | x1 = 18
                  | x2 = 10
                  | x3 = 58.29
                  | x4 = 43.6937
--------------------------------------
```

5. Comparison with metaheuristicOpt

R already offers a plethora of packages implementing metaheuristic algorithms for optimization problems. Among them, metaheuristicOpt [32], with its collection of 21 algorithms, is the most complete and popular one. Because of the clear overlap with EmiR, in this section we outline the differences between the two packages and compare their performance. Table 1 compares some of the main features offered by the two packages.

EmiR not only comes in handy to overcome the main limitation of metaheuristicOpt, that is it can only be used for continuous unconstrained optimization problems, but its faster execution times makes it appealing also in those cases where metaheuristicOpt can be used.

We compared both performance and execution time of the two packages on seven unconstrained problems, using four different algorithms: WOA, MFO, GWO and CS. The choice of these algorithms relies on the fact that the first three do not have specific configuration parameters and the fourth is implemented in a similar way in both packages, making possible to test EmiR and metaheuristicOpt in the same conditions. Each test was performed 200 times on a six-core Intel Core i7-8750H @2.20 GHz PC, running R v4.1.0 and the latest available versions (at the time of writing) of the two packages: EmiR v1.0.1 and metaheuristicOpt v2.0.0.

Table 2 summarizes the results of all tests, by reporting mean (μ) and standard deviation (σ) of both execution time and best cost of the objective function, as obtained from both packages. Values in red, in columns μ_{cost} and μ_{time}, indicate they are better than the corresponding ones from the other package. In basically all cases, EmiR performed better (sometimes much better) than metaheuristicOpt in both execution time and best cost.

6. Impact

EmiR offers a complete solution in R for solving constrained and unconstrained optimization problems. The package has been designed to be highly time effective and user-friendly, sharing a common interface to all the available algorithms implemented. Moreover, EmiR also offers a high-level of customization, as well as a set of graphical functions for the visualization of the algorithm evolution during the optimization process.

Table 1
Comparison of some of the main features of EmiR and metaheuristicOpt.

	EmiR v1.0.1	metaheuristicOpt v2.0.0
Core language	C++	R
Available algorithms	12	21
Predefined test functions	10	4
Continuous optimization	✓	✓
Integer and mixed-integer optimization	✓	–
Constrained optimization	Inequality constraints	–
Multiple options for out-of-boundary solutions	✓	–

Table 2
Performance comparison between EmiR and metaheuristicOpt tested with seven benchmark functions and four algorithms, as described in the text. The first column reports name and domain of the benchmark functions, as well as the population size and the number of iterations. Values in red, in columns μ_{cost} and μ_{time}, indicate they are better than the corresponding ones from the other package.

Function	Algo	EmiR				metaheuristicOpt			
		μ_{cost}	σ_{cost}	μ_{time}	σ_{time}	μ_{cost}	σ_{cost}	μ_{time}	σ_{time}
Ackley $x_i \in [-32.768, 32.768]$ for all $i = 1, \dots, 50$ population: 50 iterations: 2000	WOA	**3.3e−15**	2.4e−15	**2.36**	0.089	3.5e15	2.4e−15	5.85	0.19
	MFO	**12.04**	1.64	**3.53**	0.10	18.22	3.20	13.68	0.26
	GWO	1.4e−14	2.2e−15	**2.17**	0.08	**1.3e−14**	2.4e−15	56.95	0.48
	CS	**18.99**	0.20	**1.24**	0.093	20.20	0.12	3.06	0.10
Styblinski $x_i \in [-5, 5]$ for all $i = 1, \dots, 5$ population: 50 iterations: 5000	WOA	−195.83	6.1e−08	**4.04**	0.11	−195.83	3.2e−07	5.16	0.19
	MFO	**−195.80**	0.26	**4.27**	0.10	−192.01	7.19	5.98	0.20
	GWO	**−195.83**	4.6e−07	**4.06**	0.098	−193.22	6.03	18.98	0.35
	CS	**−195.83**	7.3e−05	**2.28**	0.12	−189.53	2.84	3.58	0.25
Freudenstein $x_i \in [-10, 10]$ for $i = 1, 2$ population: 50 iterations: 5000	WOA	**3.3e−06**	8.9e−06	**3.56**	0.13	9.0e−06	2.1e−05	3.76	0.18
	MFO	**0.006**	0.064	3.61	0.10	1.04	6.58	**3.54**	0.15
	GWO	**8.22**	18.53	**3.55**	0.11	9.71	19.78	8.64	0.23
	CS	**2.1e−06**	2.4e−06	**1.89**	0.043	0.052	0.055	2.28	0.062
Rastrigin $x_i \in [-5.12, 5.12]$ for $i = 1, 2$ population: 50 iterations: 10000	WOA	0.00	0.00	**6.99**	0.13	0.00	0.00	7.38	0.12
	MFO	**0.0071**	0.039	7.22	0.14	0.06	0.24	**6.24**	0.11
	GWO	0.00	0.00	**6.96**	0.094	0.00	0.00	14.57	0.34
	CS	**3.2e−06**	3.3e−06	**3.81**	0.078	0.025	0.027	4.88	0.12
Schwefel $x_i \in [-500, 500]$ for $i = 1, \dots, 4$ population: 50 iterations: 10000	WOA	**−1673.56**	16.62	**7.31**	0.15	−1643.36	82.21	8.45	0.26
	MFO	**−1630.96**	66.03	**7.63**	0.34	−1454.77	152.88	8.89	0.17
	GWO	**−1633.79**	110.85	**7.22**	0.094	−1502.06	132.18	25.95	0.29
	CS	**−1651.73**	43.24	**3.96**	0.085	−1594.24	39.21	5.50	0.10
Powell $x_i \in [-4, 5]$ for $i = 1, \dots, 20$ population: 50 iterations: 5000	WOA	**2.4e−07**	3.3e−07	**24.85**	1.77	2.7-e07	4.6e−07	29.62	1.03
	MFO	**0.88**	1.50	**25.17**	0.93	265.63	452.78	37.00	1.40
	GWO	1.9e−07	2.7e−07	**24.87**	1.73	**1.4e−07**	2.5e−07	92.57	5.94
	CS	**0.84**	0.77	**13.62**	0.72	1290.14	199.58	49.17	1.33
Sphere $x_i \in [-100, 0]$ for $i = 1, \dots, 200$ population: 10 iterations: 1000	WOA	2.0e−89	3.7e−86	**0.25**	0.016	**3.4e−156**	4.7e−142	2.64	0.06
	MFO	**2.5e+05**	2.8e+04	**1.81**	0.16	3.2e+05	2.6e+04	7.68	0.37
	GWO	3.3e−13	2.1e−13	**0.26**	0.014	**2.2e−28**	4.3e−28	30.62	1.18
	CS	**5.0e+05**	1.3e+04	**0.18**	0.019	5.1e+05	1.5e+04	1.77	0.06

From the comparison with metaheuristicOpt, which is another package for R with a large collection of metaheuristic algorithms for optimization problems, EmiR stands out for the following features:

- core code written in C++ for speed;
- possibility to handle constrained optimization problems;
- possibility to handle integer and mixed-integer optimization problems;
- different options to handle out-of-boundary solutions;
- possibility to import the initial state of the population;
- tools for plots and animations.

Although a comprehensive comparison between metaheuristic and classical optimization algorithms goes beyond the scope of this work, we also compared the performance of EmiR to some of the most used classical optimization techniques implemented in R: BFGS-method [25] (BFGS), Conjugate Gradient method (CG) [33] and Nelder Mead method (NM) [34]. Table 3 shows the average best cost (from 100 independent tests), for 5 benchmark functions,[1] as obtained from BFGS, CG and NM and three metaheuristic algorithms from EmiR: GA, WOA and ABC. To make the comparison fair, the population size and the number of iterations for each metaheuristic algorithm in this test have been chosen to target a computational time of $\mathcal{O}(1)$ second. The results clearly show that metaheuristic algorithms outperformed traditional approaches for the selected objective functions.

7. Conclusions

In this work we presented EmiR, a new package for R implementing several population-based metaheuristic algorithms for optimization problems. We started introducing its architecture,

[1] Definitions of the benchmark functions can be found at Ref. [35].

Table 3
Performance comparison between traditional (BFGS, CG, NM) and metaheuristic (GA, WOA, BAT) algorithms tested with five benchmark functions, as described in the text. The first column reports name and domain of the benchmark functions.

Function	Algorithm	μ_{cost}
Hartmann $x_i \in [0, 1]$ for all $i = 1, 2, 3$ $f_{min}(\mathbf{x}) = -3.86278$	BFGS	−3.05
	CG	−2.89
	NM	−2.65
	GA	−3.86
	WOA	−3.86
	ABC	−3.86
Michalewicz $x_i \in [0, \pi]$ for all $i = 1, \ldots, 5$ $f_{min}(\mathbf{x}) = -4.687658$	BFGS	−1.85
	CG	−2.05
	NM	−2.53
	GA	−4.68
	WOA	−4.60
	ABC	−4.68
Schwefel $x_i \in [-500, 500]$ for all $i = 1, \ldots, 5$ $f_{min}(\mathbf{x}) = -2094.9145$	BFGS	−1106.62
	CG	−1125.84
	NM	−1095.08
	GA	−2094.91
	WOA	−2046.94
	ABC	−2094.91
Schubert $x_i \in [-10, 10]$ for all $i = 1, 2$ $f_{min}(\mathbf{x}) = -186.7309$	BFGS	1.83e−18
	CG	1.54e−15
	NM	−50.38
	GA	−186.73
	WOA	−186.73
	ABC	−186.73
Styblinski–Tang $x_i \in [-5, 5]$ for all $i = 1, \ldots, 15$ $f_{min}(\mathbf{x}) = -587.48985$	BFGS	−479.34
	CG	−477.22
	NM	−460.77
	GA	−587.32
	WOA	−587.48
	ABC	−587.48

based on a common interface to all algorithms, and then we showed its usage and performance by means of several representative examples of unconstrained and constrained optimization problems, with both continuous and integer variables. We also gave an overview of some of graphical tools available in EmiR: from the graph of the best cost as function of the iteration, to the production of animated gif of the population motion. In the second part of this work, we focused on comparing the performance of EmiR with metaheuristicOpt (another package with metaheuristic algorithms for optimization problems), as well as with classical optimization methods. Tests were performed using challenging benchmark functions, with many local minima and/or high dimensionality, and results have proven the effectiveness of EmiR in terms of computational time and convergence to the global minimum. Future version of EmiR will include the addition of new algorithms and the extension of the current graphical tools to high dimensionality functions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Yang X-S. Nature-inspired metaheuristic algorithms. Luniver Press; 2010.
[2] Simon D. Evolutionary optimization algorithms. Chichester: Wiley-Blackwell; 2013.
[3] Glover F. Future paths for integer programming and links to artificial intelligence. Comput Oper Res 1986;13(5):533–49. http://dx.doi.org/10.1016/0305-0548(86)90048-1.
[4] Hussain K, Salleh MNM, Cheng S, Shi Y. Metaheuristic research: a comprehensive survey. Artif Intell Rev 2018;52(4):2191–233. http://dx.doi.org/10.1007/s10462-017-9605-z.
[5] Glover F, Sörensen K. Metaheuristics. Scholarpedia 2015;10(4):6532. http://dx.doi.org/10.4249/scholarpedia.6532, revision #149834.
[6] Karaboga D, Basturk B. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 2007;39(3):459–71. http://dx.doi.org/10.1007/s10898-007-9149-x.
[7] Tereshko V, Loengarov A. Collective decision making in honey-bee foraging dynamics. Comput Inf Syst 2005;9(3):1.
[8] Yang X-S. A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization. Springer; 2010, p. 65–74.
[9] Yang X-S, Deb S. Cuckoo search via lèvy flights. In: 2009 world congress on nature & biologically inspired computing. IEEE; 2009, http://dx.doi.org/10.1109/nabic.2009.5393690.
[10] Holland JH. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. Cambridge, MA, USA: MIT Press; 1992.
[11] Goldberg D. Genetic algorithms in search, optimization, and machine learning. Reading, Mass: Addison-Wesley Publishing Company; 1989.
[12] Rashedi E, Nezamabadi-pour H, Saryazdi S. GSA: A gravitational search algorithm. Inform Sci 2009;179(13):2232–48. http://dx.doi.org/10.1016/j.ins.2009.03.004.
[13] Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. Adv Eng Softw 2014;69:46–61. http://dx.doi.org/10.1016/j.advengsoft.2013.12.007.
[14] Muro C, Escobedo R, Spector L, Coppinger R. Wolf-pack *(Canis lupus)* hunting strategies emerge from simple rules in computational simulations. Behav Process 2011;88(3):192–7. http://dx.doi.org/10.1016/j.beproc.2011.09.006.
[15] Geem ZW, Kim JH, Loganathan G. A new heuristic optimization algorithm: Harmony search. Simulation 2001;76(2):60–8. http://dx.doi.org/10.1177/003754970107600201.
[16] Lee KS, Geem ZW. A new structural optimization method based on the harmony search algorithm. Comput Struct 2004;82(9–10):781–98. http://dx.doi.org/10.1016/j.compstruc.2004.01.002.
[17] Mahdavi M, Fesanghary M, Damangir E. An improved harmony search algorithm for solving optimization problems. Appl Math Comput 2007;188(2):1567–79. http://dx.doi.org/10.1016/j.amc.2006.11.033.
[18] Mirjalili S. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. Knowl-Based Syst 2015;89:228–49. http://dx.doi.org/10.1016/j.knosys.2015.07.006.
[19] Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of ICNN'95 - international conference on neural networks, vol. 4. 1995, p. 1942–8 vol.4. http://dx.doi.org/10.1109/ICNN.1995.488968.

[20] Kennedy J, Eberhart RC. Swarm intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2001.
[21] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220(4598):671–80. http://dx.doi.org/10.1126/science.220.4598.671.
[22] Corana A, Marchesi M, Martini C, Ridella S. Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm. ACM Trans Math Softw 1987;13(3):262–80. http://dx.doi.org/10.1145/29380.29864.
[23] Askarzadeh A, dos Santos Coelho L, Klein CE, Mariani VC. A population-based simulated annealing algorithm for global optimization. In: 2016 IEEE international conference on systems, man, and cybernetics. SMC, IEEE; 2016, http://dx.doi.org/10.1109/smc.2016.7844961.
[24] Mirjalili S, Lewis A. The whale optimization algorithm. Adv Eng Softw 2016;95:51–67. http://dx.doi.org/10.1016/j.advengsoft.2016.01.008.
[25] Haupt R. Practical genetic algorithms. Hoboken, N.J: John Wiley; 2004.
[26] Cragg E, Levy A. Study on a supermemory gradient method for the minimization of functions. J Optim Theory Appl 1969;4(3):191–205.
[27] Ackley DH. A connectionist machine for genetic hillclimbing. Springer US; 1987, http://dx.doi.org/10.1007/978-1-4613-1997-9.
[28] Ng C-K, Zhang L-S, Li D, Tian W-W. Discrete filled function method for discrete global optimization. Comput Optim Appl 2005;31(1):87–115.
[29] Reid DJ. Genetic algorithms in constrained optimization. Math Comput Modelling 1996;23(5):87–111.
[30] Liang J, Runarsson T, Mezura-Montes E, Clerc M, Suganthan P, Coello CC, Deb K. Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. 2005.
[31] Kazemzadeh Azad S, Hasançebi O, Erol OKa. Evaluating efficiency of big-bang big-crunch algorithm in benchmark engineering optimization problems. Int J Optim Civ Eng 2011;1(3). http://arxiv.org/abs/http://ijoce.iust.ac.ir/article-1-53-en.pdf, URL http://ijoce.iust.ac.ir/article-1-53-en.html.
[32] Septem Riza L, Iip, Prasetyo Nugroho E, Adi Prabowo MB, Junaeti E, Abdullah AG. Metaheuristicopt: Metaheuristic for optimization. 2019, URL https://CRAN.R-project.org/package=metaheuristicOpt, R package version 2.0.0.
[33] Hestenes M, Stiefel E. Methods of conjugate gradients for solving linear systems. J Res Natl Bur Stand 1952;49(6):409. http://dx.doi.org/10.6028/jres.049.044.
[34] Lagarias JC, Reeds JA, Wright MH, Wright PE. Convergence properties of the Nelder–Mead simplex method in low dimensions. SIAM J Optim 1998;9(1):112–47.
[35] Surjanovic S, Bingham D. Virtual library of simulation experiments: Test functions and datasets, Retrieved March 21, 2022, from http://www.sfu.ca/~ssurjano.