Journal of Engineering Science and Technology
Vol. 16, No. 2 (2021) 1006 - 1018

DEVELOPMENT AND EXPERIMENTATION OF R PACKAGE "metaheuristicOpt" ON CONTINUOUS OPTIMIZATION

LALA SEPTEM RIZA[1,*], MUHAMMAD BIMA ADI PRABOWO[1], ENJUN JUNAETI[1], ADE GAFAR ABDULLAH[1], KHYRINA AIRIN FARIZA[2]

[1]Universitas Pendidikan Indonesia Jl. Dr. Setiabudi no 229, Bandung 40154, Indonesia
[2] Universiti Teknologi Mara Cawangan Melaka Kampus Jasin Melaka, Malaysia
*Corresponding Author: lala.s.riza@upi.edu

Abstract

Optimization is applied in such various disciplines as civil engineering, mechanical engineering, economics, electrical engineering, and so on so that it plays an important role in human beings' life. There are so many approaches implemented in optimization; one of which is population-based metaheuristics. In the R programming language, there is an optimization package using a population-based metaheuristic algorithm namely "metaheuristicOpt". However, the algorithms have two drawbacks, including high complexity and few hyperparameter. The purpose of this research is to develop the R metaheuristicOpt package by adding 10 new algorithms namely clonal selection algorithm, differential evolution, shuffled frog leaping, cat swarm optimization, artificial bee colony algorithm, krill herd algorithm, cuckoo search, bat algorithm, gravitational based search, and black hole optimization to cover up the weaknesses of the previous algorithms. In adding these algorithms, we maintain the consistency of the package architecture. To analyse the performance of the added algorithms, each added algorithm is tested using 13 test functions. The benchmarks of the experiment are fitness and execution time. Based on experiments conducted, several added algorithms have a faster execution speed in comparison with the previous algorithms and some added algorithms also have better fitness than the previous algorithms.

Keywords: Metaheuristic algorithm, Optimization, R programming language, Software library, Swarm intelligence.

1. Introduction

Optimization is the process of choosing the best solution out of a collection of candidate solutions. Optimization is applied in various scientific fields such as mechanics for optimizing robot motion (rigid body dynamic) [1]; economics for optimizing economic decision making based on time [2]; electrical engineering in electro active filter optimization [3]; and civil engineering in optimizing reservoir expenditure [4]. Optimization algorithms are divided into two kinds of approaches, namely exact and approximation [5]. The exact approach always provides the best solution but has a slow computing speed. In the meantime, the approximation approach does not always provide the best solution but has a better computational speed than the exact approach. In the approximation approach, there is one branch, namely metaheuristics.

Metaheuristics consists of two words, meta which means high level and heuristic which means approach [6]. It can be concluded that metaheuristics is a high-level approach algorithm. It is called an approach because it does not provide a definite solution and is called a high level because it can solve the laxity by other approximation algorithms, which are easily trapped locally optima. Metaheuristic algorithms can be subdivided into two parts; single-based and population-based [5]. Single-based metaheuristics evaluates one candidate solution then moves using local search. Population-based metaheuristics, on the other hand, evaluates several candidate solutions and then moves based on other candidate solutions. There are lots of optimization software libraries that use population-based metaheuristic algorithms. For instance, in the Python programming language, there is a "NiaPy" package [7], C which has a "LibOPT" package [8] and Matlab via the "global optimization toolbox" [9].

There are also many packages in the R programming language optimization using population-based metaheuristic algorithms such as "DEoptim" [10] using differential evolution algorithm, "hydroPSO" [11] using the particle swarm optimization algorithm, "microbats" [12] using the bat algorithm and "ABCoptim" [13] using artificial bee colony algorithm. Some of the weaknesses of those packages are that they have different inputs and outputs, and that each algorithm has its own package. Users must prepare different inputs and after that the output must be reprocessed before being analysed because it has a different format. Whereas, in certain cases, R users must use several algorithms. For example, when the optimization process is carried out, it is highly recommended to use several algorithms to get maximum optimization results. The reason is most likely to be "No Free Lunch Theorem" [14] which says there is no optimization algorithm able to solve all optimization problems perfectly.

To solve this problem, there is an R package "metaheuristicOpt" [15] applying 11 population-based metaheuristic algorithms. It has Particle Swarm Optimization [16], Ant Lion Optimizer [17], Gray Wolf Optimizer [18], Dragonfly Algorithm [19], Firefly Algorithm [20], Genetic Algorithms [21], Grasshoper Optimization Algorithms [22], Moth Flame Optimizer [23], Sine Cosine Algorithm [24], Whale Optimization Algorithm [25] and Harmony Search [26]. The input and output of each algorithm in the "metaheuristicOpt" package has the same format (consistent). Moreover, there is a main function in which it is able to call several algorithms with one input so that users do not need to call several algorithms with the same input.

Package “metaheuristicOpt” only contains 11 population-based metaheuristic algorithms. It has been proven that from 1975 to 2012, there are approximately 31 metaheuristic algorithms [6]. There is a high chance that other algorithms outside the 11 algorithms owned by the "metaheuristicOpt" package can perform better optimization on certain problems. In addition, some algorithms in the R metaheuristicOpt package have a high complexity. High complexity causes the higher the iteration the slower the algorithm. Algorithms that have high complexity in the R package "metaheuristicOpt" include ant lion optimizer, dragonfly algorithm, gray wolf optimizer, firefly algorithm, moth flame optimizer, sine cosine algorithm and grasshooper optimization algorithm. Most of the algorithms in the "metaheuristicOpt" R package have few hyperparameters. This means that if the algorithm performs an optimization on an optimization problem, it gets a high error (far from global optima) then the algorithm has no way of reducing the error. However, if an algorithm has a lot of hyperparameters when optimizing, it gets high error then the error can be reduced or even eliminated by tuning hyperparameter. Algorithms that have few hyperparameters in the R metaheuristicOpt package include ant lion optimizer, dragonfly algorithm, gray wolf optimizer, grasshopper optimization algorithm, moth flame optimizer, sine cosine algorithm and whale optimizer algorithm.

The purpose of this research is to develop a "metaheuristicOpt" R package by adding 10 algorithms namely Clonal Selection Algorithm [27], Differential Evolution [28], Shuffled Frog Leaping [29], Cat Swarm Optimization [30], Artificial Bee Colony Algorithm [31] , Krill Herd Algorithm [32], Cuckoo Search [33], Bat Algorithm [34], Gravitational Based Search [35] and Black Hole Optimization [36]. The addition of 10 algorithms is to cover the weaknesses of the algorithm owned by the previous "metaheuristicOpt" R package. Algorithms that have little complexity include differential evolution, artificial bee colony algorithm, cuckoo search, bat algorithm and black hole optimization. Algorithms that have many hyperparameters include clonal selection algorithm, differential evolution, shuffled frog leaping, paint swarm optimization, krill herd algorithm and gravitational based search. The addition of 10 algorithms is expected to increase the variance of the "metaheuristicOpt" R package. Researchers chose to develop this package rather than creating a new package because the "metaheuristicOpt" package is created with the concept of consistency of input and output. In addition, "metaheuristicOpt" is made using the R programming language which is widely used by data used by scientists who really need optimization [37].

2. Proposed Methods

2.1. Optimization using population-based metaheuristic algorithm

Optimization is the process of finding the best solution from a set of solutions. Optimization is synonymous with the word maximal, optimal, ideal, etc.

Broadly speaking, the population-based metaheuristic algorithm has 3 main components (see Fig. 1): (i) Initializing several candidate solutions (population) randomly; (ii) Updating the population using some rules as many as iteration; and (iii) Selecting the best candidate solution from the population as an algorithm result. Moreover, the "metaheuristicOpt" R package uses 11 population-based metaheuristic algorithms, namely: (1) Particle Swarm Optimization [16]; this algorithm is inspired by the behaviour of individuals from fish herds or groups of birds that fly together in

search of food or nests; (2) Ant Lion Optimizer [17]; this algorithm is inspired by the behaviour of the antlion and its Latin name is Myrmeleontidae, which means hunting ants; (3) Grey Wolf Optimizer [18]; this algorithm is inspired by the hunting behaviour of gray wolves (Canis lupus); (4) Dragonfly Algorithm [19]; this algorithm is inspired by the behaviour of swarm of dragonflies; (5) Firefly Algorithm [20]; this algorithm is inspired by the flickering behaviour of fireflies; (6) Genetic Algorithm [21]; this algorithm is inspired by genetic behaviour and natural selection; (7) Grasshoper Optimization Algorithm [22]; this algorithm is inspired by the behaviour of group grasshoppers in nature; (8) Moth Flame Optimizer [23]; this algorithm is inspired by the behaviour of flying moths around a light source; (9) Sine Cosine Algorithm [24]; as the name implies, this algorithm is inspired by the characteristics of sine and cosine functions; (10) Whale Optimization Algorithm [25]; this algorithm is inspired by the behaviour of humpback whales (humpback) and its Latin name is Megaptera novaeangliae, which means looking for food; and (11) Harmony Search [26]; this algorithm is inspired by the improvisation process of musicians in finding suitable tone combinations.

```
Input : objective function f(x)

Output : best solution

Initialize population P0

Evaluate each candidate solution at P0

while t < max Iteration

   update position each candidate solution P0

   evaluate new candidate solution

   t = t+1

end while
```

Fig. 1. Template of population based metaheuristic algorithm [15].

We improve R package "metaheuristicOpt" by adding 10 population-based metaheuristic algorithms, they are (1) Clonal Selection Algorithm [27] which is inspired by the maturation process of the human immune system; (2) Differential Evolution [28] using evolutionary stages such as mutation, crossover and selection for optimization; (3) Shuffled Frog Leaping [29] which is inspired by the behaviour of frogs in search of food; (4) Cat Swarm Optimization [30] which is inspired by feline behaviour (cats, tigers, lions) when trying to find prey and rest; (5) Artificial Bee Colony Algorithm [31] which is inspired by the division of work carried out by swarms of bees; (6) Krill-Herd Algorithm [32] which is inspired by the behaviour of the herd of krill against the surrounding environment, other krill and food; (7) Cuckoo Search [33] which is inspired by the behaviour of cuckoo birds when laying eggs; (8) Bat Algorithm [34] which inspired by bat's ecolocation ability; (9) Gravitational Based Search [35] using the first Newton law (law of universal gravity) to do optimization; and (10) Black Hole Optimization [36] which is inspired by the behaviour of black holes that suck stars around it.

2.2. R package "metaheuristicOpt" architecture

The "metaheuristicOpt" package R implements 21 population-based metaheuristic algorithms. This package is stored on the Comprehensive R Archive Network (CRAN) server. The specifications of this package can be seen on the page https://cran.r-project.org/web/packages/metaheuristicOpt/index.html. Users can download this package for free.

Figure 2 is the package architecture after the addition of a new algorithm. Each algorithm has a function with an acronym for the name of the algorithm. We call them algorithm function. The algorithm functions include `CLONALG()`, `DE()`, `SFL()`, `CSO()`, `ABC()`, `KH()`, `CS()`, `BA()`, `GBS()`, `BHO()`, `PSO()`, `ALO()`, `GWO()`, `DA()`, `FA()`, `GA()`, `GOA()`, `MFO()`, `SCA()`, `WOA()` and `HS()`.

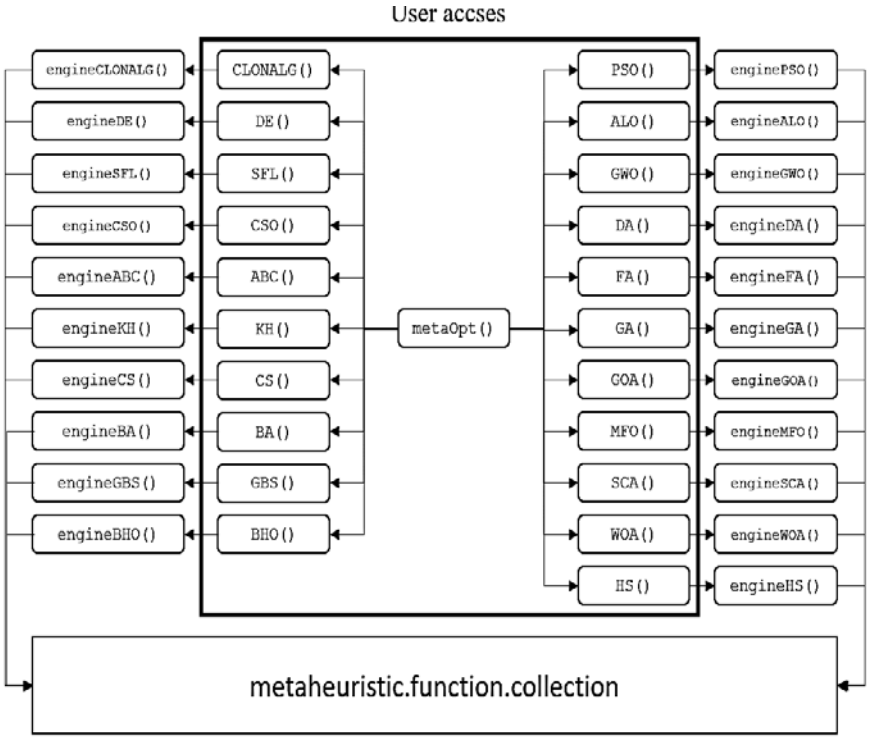

Fig. 2. R package "metaheuristicOpt" architecture.

The algorithm function can be accessed by users. Each algorithm functions as input validation, stores default values and calls engine functions. The algorithm function parameters have at least general parameters such as:

- `FUN`: determine the objective function with *Function* as the data type.
- `numVar`: determines many input and output variables with *numeric* as the data type.
- `numPopulation`: determine how much population with *numeric* as the data type.
- `maxIter`: specify many iterations with *numeric* as the data type.
- `rangeVar`: specify the algorithm search limit with *matrix* as the data type.

However, special parameters are determined by themselves.

Engine function is a function that runs based on a predetermined algorithm. The engine functions include `engineCLONALG()`, `engineDE()`, `engineSFL()`, `engineCSO()`, `engineABC()`, `engineKH()`, `engineCS()`, `engineBA()`, `engineGBS()`, `engineBHO()`, `enginePSO()`, `engineALO()`, `engineGWO()`, `engineDA()`, `engineFA()`, `engineGA()`, `engineGOA()`, `engineMFO()`, `engineSCA()`, `engineWOA()` and `engineHS()`. The engine function cannot be accessed by users, yet they are accessible through the algorithm function. For functions that are commonly used in population-based metaheuristic algorithms such as initializing random candidate solutions, calculating fitness and selecting the best candidate solution, the engine function will access `metaheuristic.function.collection`. `metaheuristic.function.collection` which are not accessible by users.

To make it easier for users, we create `metaOpt()` generalization functions. The `metaOpt()` function can be accessed by the users. These functions enable them to call several algorithm functions using the same input. Using the `metaOpt()` function, users do not need to call the algorithm function many times for the same input.

2.3. Test function

The Test Function is a function used to measure the performance and performance of an optimization algorithm. Some test functions are very easy because they have one local optima which is at the same time global optima (unimodal) and some are quite difficult because they have several local optima that function as traps (multimodal) [38]. Test functions unimodal are sphere model ($\mathbf{f_1}$), Schwefel's problem 2.22 ($\mathbf{f_2}$), Schwefel's problem 1.2 ($\mathbf{f_3}$), Schwefel's problem 2.21 ($\mathbf{f_4}$), generalized Rosenbrock's ($\mathbf{f_5}$), step function ($\mathbf{f_6}$), and quartic function with noise ($\mathbf{f_7}$). Test functions multimodal are generalized Schwefel's problem 2.26 ($\mathbf{f_8}$), generalized Rastrigin's function ($\mathbf{f_9}$), Ackley's function ($\mathbf{f_{10}}$), generalized Griewank function ($\mathbf{f_{11}}$), generalized penalized function 1 ($\mathbf{f_{12}}$), and generalized penalized function 2 ($\mathbf{f_{13}}$).

3. Methods

Special parameters (hyperparameter) use the default parameters while the general parameters (except `rangeVar`) use a combination of the following:

- `FUN` ∈ {$\mathbf{f_1}$ to $\mathbf{f_{13}}$}
- `numVar` ∈10
- `maxIter` ∈ {100, 500, 1000}
- `numPopulation` ∈ {10, 30, 50, 100}

Overall, each function is tested for as many as 156 combinations. The general parameters for rangeVar are filled with range of the test function. The test function (`FUN`) that we use is based on a previous study [38].

We also analyse the optimum value (fitness) and the time required. An algorithm function has good performance if fitness is close to global optima or right on the global optima of the test function.

4. Results and Discussion

This chapter explains the results and analysis of experiments. Table 1 explains the most optimum value obtained from each new function (the newly added algorithm) which is added when the `maxIter` parameter is 500 and `numPopulation` is 30.

Table 1. Optimum value of new functions.

	CLONALG	DE	SFL	CSO	ABC	KH	CS	BA	GBS	BHO	Global Optima
f_1	0,418021204	2,65E-17	7,24E-11	2,62E-26	5,18E-09	341,8820472	4563,749985	**0**	8,71E-05	5,14E-12	**0**
f_2	0,34975769	6,65E-11	9,95E-06	9,13E-16	9,21E-06	12,54653071	22,98476484	**0**	0,001408983	3,41E-08	**0**
f_3	292,0115131	1,095095976	0,535062919	1,34E-22	1087,28759	2484,60606	6108,85962	**0**	0,000111291	7,994177915	**0**
f_4	1,525227726	0,00053144	13,41466134	7,18E-14	3,740859322	10,27965295	34,27125011	**0**	0,004718069	0,005741752	**0**
f_5	15,48559859	3,779471314	5,564553864	8,050803055	2,609815807	96411,18483	2870621,151	8,846584374	**1,13495026****7**	7,983755995	**0**
f_6	0,113926231	**1,44E-17**	1,31E-07	0,011210143	1,33E-08	177,878059	6520,920573	2,002835664	4,82E-05	1,23E-08	**0**
f_7	0,600775694	0,212384544	**0,03149202****9**	0,070698452	0,924950335	2,025566489	0,655653489	0,452625232	0,947047536	0,755534083	**0**
f_8	**-4188,186033**	-5,92E+58	-2920,134709	-1,95526E+18	-9927302,228	-50251,4824	-2368,483331	-2660,329774	-1761,173884	-2269,392996	**4.189,82**
f_9	0,067896281	0,994959133	4,974795361	**0**	0,432687156	26,72782968	63,9158553	**0**	5,969787962	4,975122016	**0**
f_{10}	0,456076188	2,54E-09	3,12E-06	1,35E-13	0,001511501	5,77786715	16,4762729 1	**4,44E-16**	0,002979477	2,06E-07	**0**
f_{11}	0,272467718	0,027157363	0,124107869	**0**	0,000130688	4,552391873	44,02092439	**0**	0,04204186	0,177310506	**0**
f_{12}	0,003335408	**9,72E-19**	3,35E-12	0,015168471	4,77E-10	22,59959641	1050926,93	0,334585143	1,868111762	7,92E-09	**0**
f_{13}	0,007086916	9,26E-18	9,17E-06	6,54E-11	2,08E-08	33046,68461	18092736,94	0,913883861	1,78E-06	8,82E-13	0

It can be seen that the `BA()` function has the best optimum value in 7 test functions ($\mathbf{f_1}$ through $\mathbf{f_4}$ and $\mathbf{f_9}$ through $\mathbf{f_{11}}$). It should be noted that all function tests run well where `BA()` gets the most optimum value with an optimum variable value of 0. The best fitness $\mathbf{f_9}$ and $\mathbf{f_{11}}$ are also achieved by the `CSO()` function. The `DE()` function succeeds in getting the best fitness on 3 test functions ($\mathbf{f_6}$, $\mathbf{f_{12}}$ and $\mathbf{f_{13}}$). For other test functions, the best fitness is $\mathbf{f_5}$ by `GBS()`, $\mathbf{f_7}$ by `SFL()` and $\mathbf{f_8}$ by `ClONALG()`. Although there are some functions such as `ABC()`, `KH()`, `CS()` and `BHO()` that do not get the best fitness, they are still considered successful because it is possible that the optimization algorithm can get better fitness on the test function outside the 13 test functions used on this experiment.

Table 2 explains the most optimum values obtained from each of the old functions (functions that already exist in the previous R "mertaheuristicOpt" package) when the maxIter parameter is 500 and numPopulation is 30.

It can be seen in Table 2 that the algorithm with the best performance is dominated by `WOA()` in $\mathbf{f_1}$, $\mathbf{f_2}$ and $\mathbf{f_8}$ through $\mathbf{f_{10}}$. `GWO()` is then proven to have the best performance on $\mathbf{f_3}$, $\mathbf{f_4}$, $\mathbf{f_9}$ and $\mathbf{f_{11}}$. In addition, `PSO()` is likely to have best performance on $\mathbf{f_5}$, $\mathbf{f_6}$, $\mathbf{f_{12}}$ and $\mathbf{f_{13}}$ and finally, `HS()` has the best performance on $\mathbf{f_7}$ and $\mathbf{f_8}$. Furthermore, new and old functions with the best fitness are selected and compared on Table 3.

Table 2. Optimum value of previous algorithms.

	PSO	ALO	GWO	DA	FFA	GA	GOA	MFO	SCA	WOA	HS	Global Optima
f_1	7.31E-25	7.06E-09	2.84E-125	7.05E-02	6.68E+02	3.26E+00	6.38E-06	4.56E-19	5.35E-16	**2.37E-134**	3.53E-05	**0**
f_2	5.50E-14	6.68E+00	2.14E-71	2.48E+00	7.72E-01	4.22E-01	1.13E-03	3.76E-13	1.07E-08	**3.29E-75**	1.24E-02	**0**
f_3	1.80E-17	2.25E-03	**1.17E-62**	3.14E+02	1.44E+03	9.96E+02	8.81E-02	5.32E-03	1.04E-04	1.23E+02	1.78E+02	**0**
f_4	6.07E-11	3.10E+00	**1.86E-41**	2.05E+01	4.82E+01	4.52E+00	1.90E-01	7.24E+00	1.46E-05	1.98E+01	1.60E+00	**0**
f_5	**8.20E-02**	1.63E+01	7.19E+00	2.59E+03	1.43E+03	5.92E+01	2.86E+01	7.21E-01	7.37E+00	7.18E+00	2.42E-01	**0**
f_6	**3.13E-25**	2.76E-09	1.09E-06	3.80E+01	7.07E+02	9.82E+00	2.43E-06	9.04E-19	4.29E-01	7.74E-04	3.28E-05	**0**
f_7	6.56E-01	6.19E-01	7.58E-01	9.21E-01	1.03E+00	1.03E+00	9.90E-01	9.44E-01	4.94E-01	3.54E-01	**2.55E-01**	**0**
f_8	-1614.0114	-2285.1506	-2233.5237	-2728.7043	-2610.0270	-4165.2984	-2985.6876	-3479.1989	-2249.9277	-4188.7035	**-4188.6819**	**4.189,82**
f_9	6.96E+00	3.08E+01	**0.00E+00**	4.14E+01	7.10E+01	1.70E+00	6.77E+01	4.88E+01	2.56E-09	**0.00E+00**	5.34E-03	**0**
f_{10}	1.11E-14	2.58E+00	7.55E-15	3.03E+00	8.20E+00	9.90E-02	2.32E+00	2.36E-10	2.90E-08	**4.44E-16**	6.34E-03	**0**
f_{11}	8.03E-01	2.29E-01	**2.45E-02**	1.14E-01	1.20E+02	1.05E+00	2.23E-01	8.36E-02	4.51E-02	1.46E-01	1.18E-01	**0**
f_{12}	**1.21E-26**	8.73E+00	1.57E-07	4.53E-01	2.69E+01	3.93E-01	2.58E-01	1.17E-21	1.57E-01	1.92E-02	1.05E-06	**0**
f_{13}	**1.45E-27**	7.58E-08	1.54E-06	2.59E+04	2.34E+01	2.27E-01	2.11E-02	1.10E-02	5.68E-01	2.20E-02	1.60E-02	**0**

Table 3. Comparison between of new function and old function.

Test Function	best previous algorithm	Fitness of best previous algorithm	Best newly added algorithm	Fitness of Best newly added algorithm	Best algorithm
f_1	`WOA()`	2.37E-134	`BA()`	0	`BA()`(new)
f_2	`WOA()`	3.29E-75	`BA()`	0	`BA()`(new)
f_3	`GWO()`	1.17E-62	`BA()`	0	`BA()`(new)
f_4	`GWO()`	1.86E-41	`BA()`	0	`BA()`(new)
f_5	`PSO()`	8.20E-02	`GBS()`	1,134950267	`PSO()`(old)
f_6	`PSO()`	3.13E-25	`DE()`	1,44E-17	`PSO()`(old)
f_7	`HS()`	2.55E-01	`SFL()`	0,031492029	`HS()`(old)
f_8	`HS()`	-4188.6819	`CLONALG()`	-4188,18603	`CLONALG()`(new)
f_9	`GWO()`, `WOA()`	0	`CSO()` and `BA()`	0	`GWO()`, `WOA()`, `CSO()` and `BA()`
f_{10}	`WOA()`	4.44E-16	`BA()`	4,44E-16	`WOA()` and `BA()`
f_{11}	`GWO()`	2.45E-02	`CSO()`and `BA()`	0	`CSO()` and `BA()`(new)
f_{12}	`PSO()`	1.21E-26	`DE()`	9,72E-19	`PSO()`(old)
f_{13}	`PSO()`	1.45E-27	`DE()`	9,26E-18	`PSO()`(old)

Comparison between the new function and the old function can be seen in Table 3. `BA()` function still has the best fitness in 7 test functions (f_1 through f_4 and f_9 through f_{11}), while the best fitness of f_5, f_6, f_{12} and f_{13} is achieved by `PSO()` function. The best fitness of f_7 is achieved by `HS()` and f_8 is achieved by `ClONALG()`.

Analysis in terms of execution time can be seen in Figs 3 and 4. Figures 3 and 4 present the average values of the time taken from the whole experiment in each function. Figure 3 shows the average of new functions and Fig. 4 depicts average of old functions. The execution time is calculated in seconds.

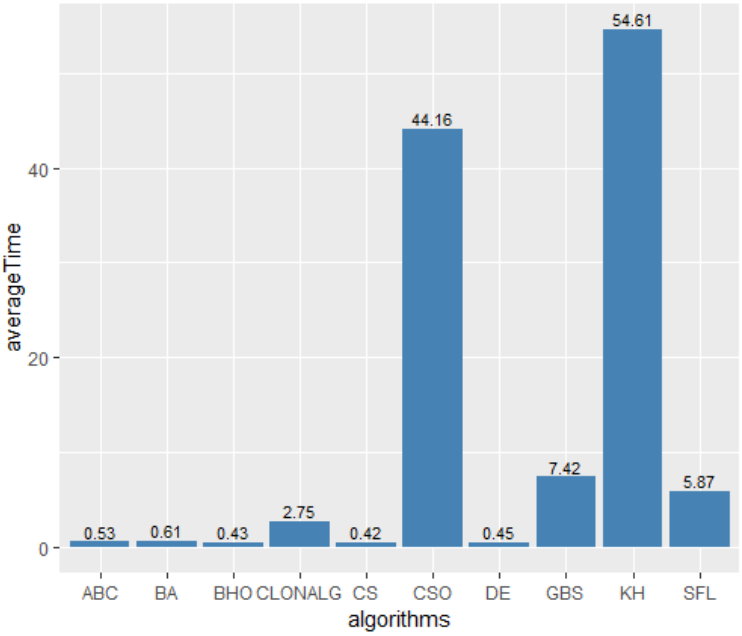

Fig. 3. Average execution time of new functions.

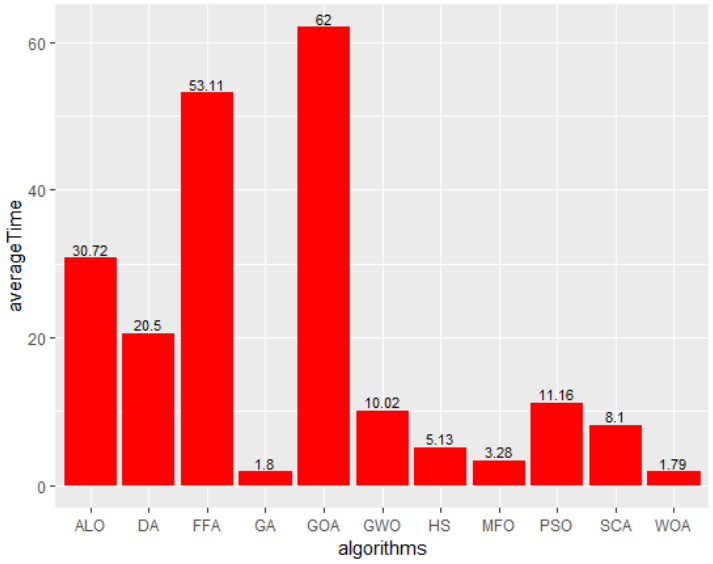

Fig. 4. Average execution time of old functions.

In Fig. 3, new functions such as `ABC()`, `BA()`, `BHO()`, `CS()` and `DE()` have an average execution time of under 1 second. However, in Fig. 4, the old function has an average execution time of above 1 second. These functions can get an average of under 1 second because they have low complexity so that vector operations are easy to carry out.

R programming languages is an interpreter programming language. The interpreter programming languages compile and run one line of code at time, unlike compiler programming languages that compile the whole codes then run it. Therefore, the use of loops must be reduced because the interpreter will compile the same line repeatedly causing the computational speed to be slow. One way to reduce the loop is to use vector operations.

Fast execution time does not always guarantee optimal value. For example, in the test function f_6. `PSO()` has slower execution time than `BA()` but gets better fitness than `BA()`. Another example is $\mathbf{f_7}$ and $\mathbf{f_8}$ whose best fitness is obtained by the `HS()` and `CLONALG()`.

Table 4. Comparison of "metaheuristicOpt" and another optimization library.

Software Library	Many Algorithm	Support Parallel computing	Programming language	references
metaheuristicOpt	21	No	R	
DEoptim	1	Yes	R	[10]
hydroPSO	5	Yes	R	[11]
Rmalschains	4	No	R	[39]
NMOF	5	Yes	R	[40]
microbats	1	No	R	[12]
ABCoptim	1	No	R	[13]
NiaPy	27	No	Python	[7]
LibOPT	24	No	C	[8]
Global Optimization Toolbox	6	Yes	Matlab	[9]

Comparison of R package "metaheuristicOpt" with other optimization packages in CRAN and other programming language libraries can be seen in Table 4. Based on Table 4, "metaheuristicOpt" has more algorithm variance than other optimization packages on R but not as much as other software libraries outside R such as R "NiaPy" and "LibOPT". Despite the weaknesses identified within the continuous optimization, it has been proven that the added algorithms overall succeeded as previous literature showed [41-43].

5. Conclusion

Based on the experimental results, the 10 algorithms added succeeds in covering up the weaknesses in the R package "metaheuristicOpt" at https://CRAN.R-project.org/package=metaheuristicOpt. There are two weaknesses of R package "metaheuristicOpt" identified namely high complexity and little hyperparameter. High complexity is overcome by the artificial bee colony algorithm, bat algorithm, differential evolution, cuckoo search, and black hole optimization. Meanwhile, hyperparameter is overcome by bat algorithm, cat swarm optimization, and clonal selection algorithm. The "metaheuristicOpt" package now has more variance in solving optimization problems. Researchers plan to improve the "metaheuristicOpt" by supporting parallel computing for further studies.

References

1. Vukcevic, D. (2018). *Extending a constrained hybrid dynamics solver for energy-optimal robot motions in the presence of static friction.* Sankt

Augustin: Hochschule Bonn-Rhein-Sieg University of Applied Sciences, Department of Computer Science.

2. Dorfman, R. (1969). An economic interpretation of optimal control theory. *The American Economic Review*, 59(5), 817-831.
3. De, B.P.; Kar, R.; Mandal, D.; and Ghoshal, S.P. (2015). Optimal selection of components value for analog active filter design using simplex particle swarm optimization. *International Journal of Machine Learning and Cybernetics*, 6(4), 621-636.
4. Ahmadianfar, I.; Adib, A.; and Salarijazi, M. (2016). Optimizing multireservoir operation: hybrid of bat algorithm and differential evolution. *Journal of Water Resources Planning and Management*, 142(2), 05015010.
5. Talbi, E.G. (2009). *Metaheuristics: from design to implementation*. New Jersey: John Wiley and Sons.
6. Beheshti, Z.; and Shamsuddin, S.M.H. (2013). A review of population-based meta-heuristic algorithms. *International Journal of Advances in Soft Computing and its Applications*, 5(1), 1-35.
7. Vrbančič, G.; Brezočnik, L.; Mlakar, U.; Fister, D.; and Fister, I. (2018). NiaPy: Python microframework for building nature-inspired algorithms. *Journal of Open Source Software*, 3(23), 613.
8. Papa, J.P.; Rosa, G.H.; Rodrigues, D.; and Yang, X.S. (2017). Libopt: An open-source platform for fast prototyping soft optimization techniques. *arXiv preprint arXiv:1704.05174*.
9. Birge, B. (2003). PSOt-a particle swarm optimization toolbox for use with Matlab. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, SIS'03 (Cat. No. 03EX706)*. Indianapolis, USA, 182-186.
10. Ardia, D.; Boudt, K.; Carl, P.; Mullen, K.; and Peterson, B.G. (2011). Differential evolution with DEoptim: an application to non-convex portfolio optimization. *The R Journal*, 3(1), 27-34.
11. Zambrano-Bigiarini, M.; and Rojas, R.; (2020). Package 'hydroPSO'. Retrieved December 5, 2020, from https://github.com/hzambran/hydroPSO.
12. Hwang, S.; and Moon, R.M. (2019). Microbats: An implementation of bat algorithm in R. Retrieved December 5, 2020, from https://rdrr.io/cran/microbats/.
13. Yon, G.V.; and Muñoz, E. (2019). ABCOptim: Implementation of Artificial Bee Colony (ABC) Optimization. Retrieved December 5, 2020, from *https://rdrr.io/cran/ABCoptim/*
14. Wolpert, D.H.; and Macready, W.G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1), 67-82.
15. Riza, L.S.; and Nugroho, E.P. (2018). MetaheuristicOpt: An R package for optimisation based on meta-heuristics algorithms. *Pertanika Journal of Science and Technology*, 26(3), 1401-1412.
16. Eberhart, R.; and Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Nagoya, Japan, 39-43.
17. Mirjalili, S. (2015). The ant lion optimizer. *Advances in engineering software*, 83, 80-98.

18. Mirjalili, S.; Mirjalili, S.M.; and Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69, 46-61.
19. Mirjalili, S. (2016). Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications*, 27(4), 1053-1073.
20. Yang, X.S. (2009). Firefly algorithms for multimodal optimization. *Proceedings of the International symposium on stochastic algorithms*. Sapporo, Japan, 169-178.
21. Goldberg, D.E.; and Holland, J.H. (1988). Genetic algorithms and machine learning. *Machine Learning*, 3, 95-99.
22. Saremi, S.; Mirjalili, S.; and Lewis, A. (2017). Grasshopper optimisation algorithm: theory and application. *Advances in Engineering Software*, 105, 30-47.
23. Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-based systems*, 89, 228-249.
24. Mirjalili, S. (2016). SCA: a sine cosine algorithm for solving optimization problems. *Knowledge-based systems*, 96, 120-133.
25. Mirjalili, S.; and Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software*, 95, 51-67.
26. Geem, Z.W.; Kim, J.H.; and Loganathan, G.V. (2001). A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2), 60-68.
27. De Castro, L.N.; and Von Zuben, F.J. (2002). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3), 239-251.
28. Das, S.; and Suganthan, P.N. (2010). Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1), 4-31.
29. Eusuff, M.; Lansey, K.; and Pasha, F. (2006). Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. *Engineering Optimization*, 38(2), 129-154.
30. Chu, S.C.; Tsai, P.W.; and Pan, J.S. (2006). Cat swarm optimization. *Proceedings of the Pacific Rim international conference on artificial intelligence*. Guilin, China, 854-858.
31. Karaboga, D.; and Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1), 108-132.
32. Gandomi, A.H.; and Alavi, A.H. (2012). Krill herd: a new bio-inspired optimization algorithm. *Communications in Nonlinear Science and Numerical Simulation*, 17(12), 4831-4845.
33. Yang, X.S.; and Deb, S. (2009). Cuckoo search via Lévy flights. Proceedings of the *2009 World Congress on Nature and Biologically Inspired Computing (NaBIC). Coimbatore, India*, 210-214.
34. Yang, X.S. (2011). Bat algorithm for multi-objective optimisation. *International Journal of Bio-Inspired Computation*, 3(5), 267-274.
35. Rashedi, E.; Nezamabadi-Pour, H.; and Saryazdi, S. (2009). GSA: A gravitational search algorithm. *Information Sciences*, 179(13), 2232-2248.
36. Hatamlou, A. (2013). Black hole: A new heuristic optimization approach for data clustering. *Information Sciences*, 222, 175-184.

37. Hayes, B. (2019) Programming Languages Most Used and Recommended by Data Scientists. Retrieved January 13, 2019, from http://businessoverbroad way. com/2019/01/13/program.
38. Yao, X.; Liu, Y.; and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2), 82-102.
39. Bergmeir, C.N.; Molina Cabrera, D.; and Benítez Sánchez, J.M. (2016). Memetic algorithms with local search chains in R: the Rmalschains package. *Journal of Statistical Software*, 75(4), 1-3.
40. Gilli, M.; Maringer, D.; and Schumann, E. (2016). *Numerical methods and optimization in finance*. Cambridge: Academic Press.
41. Egea, J.A.; Henriques, D.; Cokelaer, T.; Villaverde, A.F.; MacNamara, A.; Danciu, D.P.; Banga, J.R.; and Saez-Rodriguez, J. (2014). MEIGO: an open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics. *BMC Bioinformatics*, 15(1), 1-9.
42. Sörensen, K.; and Glover, F. (2013). Metaheuristics. *Encyclopedia of operations research and management science*, 62, 960-970.
43. Wang, H.; Wu, Z.; and Rahnamayan, S. (2011). Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems. *Soft Computing*, 15(11), 2127-2140.