Journal of Systems Architecture 139 (2023) 102871

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

MEALPY: An open-source library for latest meta-heuristic algorithms in Python

Nguyen Van Thieu [a,*], Seyedali Mirjalili [b,c,d]

[a] Faculty of Computer Science, PHENIKAA University, Yen Nghia, Ha Dong, Hanoi 12116, Viet Nam
[b] Centre for Artificial Intelligence Research and Optimization, Torrens University, Australia
[c] Yonsei Frontier Lab, Yonsei University, Seoul, Republic of Korea
[d] University Research and Innovation Center, Obuda University, Budapest, Hungary

ARTICLE INFO

Keywords:
Meta-heuristic algorithms
Nature-inspired algorithms
Swarm-based computing
Global search optimization
Optimization library
Python software

ABSTRACT

Meta-heuristic algorithms are becoming more prevalent and have been widely applied in various fields. There are numerous reasons for the success of such techniques in both science and industry, including but not limited to simplicity in search/optimization mechanisms, implementation readiness, black-box nature, and ease of use. Although the solutions obtained by such algorithms are not guaranteed to be exactly global optimal, they usually find reasonably good solutions in a reasonable time. Many algorithms have been proposed and developed in the last two decades. However, there is no library implementing meta-heuristic algorithms, which is easy to use and has a vast collection of algorithms. This paper proposes an open-source and cross-platform Python library for nature-inspired optimization algorithms called Mealpy. To propose Mealpy, we analyze the features of existing libraries for meta-heuristic algorithms. After, we propose the designation and the structure of Mealpy and validate it with a case study discussion. Compared with other libraries, our proposed Mealpy has the largest number of classical and state-of-the-art meta-heuristic algorithms, with more than 160 algorithms. Mealpy is an open-source library with well-documented code, has a simple interface, and benefits from minimum dependencies. Mealpy includes a wide range of well-known and recent meta-heuristics algorithms capable of optimizing challenge benchmark functions (e.g. CEC-2017). Mealpy can also be used for practical problems such as optimizing parameters for machine learning models. We invite the research community for widespread evaluations of this comprehensive library as a promising tool for research study and real-world optimization. The source codes, supplementary materials, and guidance is publicly available on GitHub: https://github.com/thieu1995/mealpy.

1. Introduction

Optimization can be considered as the process of finding the best optimal solution from the set of all possible solutions for a given optimization problem subject to not violating any of the constraints [1]. The process of optimizing a particular problem can be divided into several steps, including defining the problem systematically in mathematical form, determining the variables that affect the output of the problem and the necessary constraints that must be satisfied, and defining the properties of the system and searching for its states (values of variables) that give the desired properties either maximum or minimum.

Generally speaking, optimization algorithms can be divided into two categories: exact and approximate algorithms. In the former class, the design and implementation of algorithms are mostly based on mathematics such as dynamic programming, divide and conquer, branch-and-bound, and backtracking [2]. Such are very efficiency in many problems [3–6], however, they are not efficient enough when dealing with larger-scale or highly non-linear optimization problems. Since the search space increases exponentially with the problem size, exhaustive search is impractical in this problems [7].

In the latter group, approximation methods, the final solution is usually an approximate solution instead of an exact optimal solution. However, such techniques benefit from low computation cost compared to exact methods. The general form of these methods is based on an iterative approach. Traditional methods such as the Newton, Quasi-Newton, Interpolation, Gradient Descent, and Hessians matrix mainly require gradient calculation [8]. However, optimization problems these

* Correspondence to: Phenikaa University, Hanoi, Viet Nam.
E-mail addresses: thieu.nguyenvan@phenikaa-uni.edu.vn (N. Van Thieu), ali.mirjalili@gmail.com (S. Mirjalili).

https://doi.org/10.1016/j.sysarc.2023.102871
Received 14 January 2023; Received in revised form 28 March 2023; Accepted 30 March 2023
Available online 5 April 2023

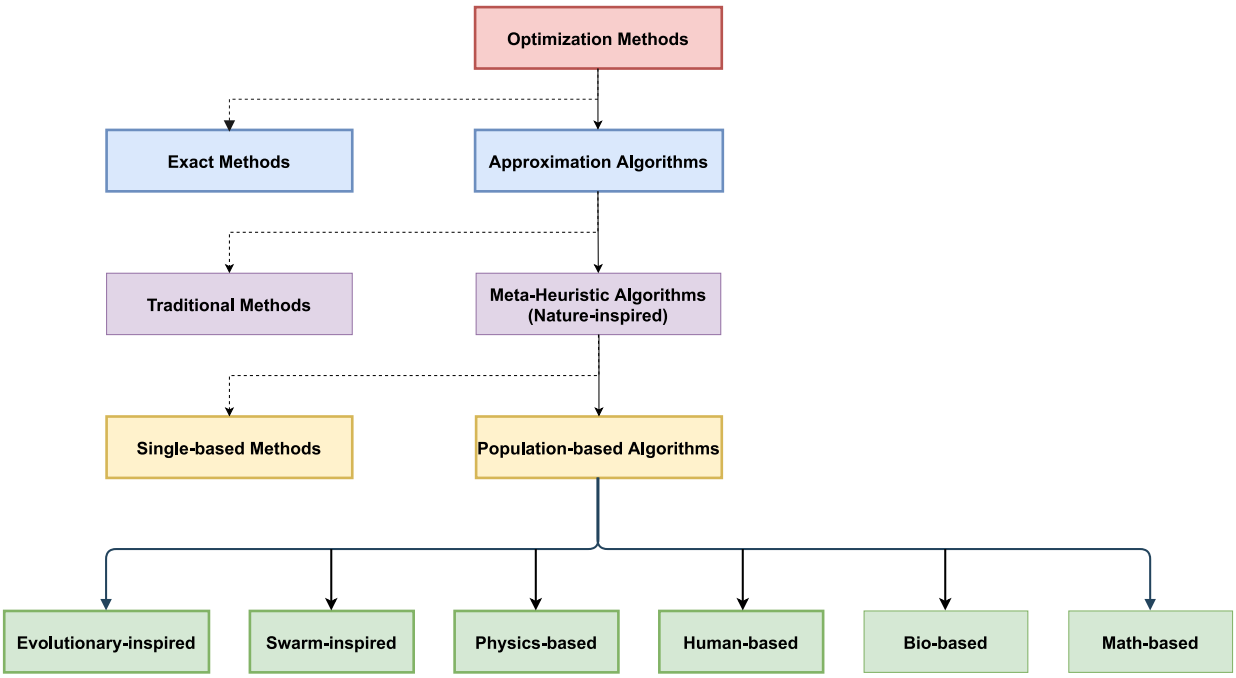


Fig. 1. Optimization methods categories.

days have become more complex in their formula and challenging to find the gradient. Some even have an implicit objective function that leads to difficulty calculating the gradient. Consequently, these limitations motivate researchers to develop more flexible and adaptable techniques.

Due to the above-mentioned reasons, Meta-heuristic algorithms (MHAs) have gradually become a primary tool of optimization. They are also approximation and iterative-based methods, but their source of inspiration usually comes from the natural behavior of organizations, systems, or a swarm of animals living in nature. This natural behavior usually forms complex rules but performs in a simple flow. Mainly, by modeling the behavior of the organizations, a new and efficient optimization method can be established. An MHA is formed by randomization and heuristic rules that draw from natural phenomena. Besides the simple idea (due to the natural characteristics of the idea taken from nature, close to humans) and ease of implementation, MHAs also can search for solutions locally and globally based on exploration and exploitation phases and can avoid the requirement of gradient information (non-domain specific), and can escape from local optima to avoid premature convergence.

In general, there are numerous ways to categorize MHAs. The popular approach is divided into a single-based solution and a population-based solution branch [9]. For example, the simulated annealing (SA) algorithm [10] is a single-based solution that takes an initial solution, updates its position during the evolution and searching phases, and eventually reaches a final state of the solution, which is considered the final solution for the problem. However, single-based solution algorithms do not receive much attention because of their disadvantages, such as simplicity and difficulty balancing exploration and exploitation when only one solution exists. In addition, it is easy to get biased by the biased initialization of the first solution. These disadvantages are completely overcome in population-based solution algorithms, in which a set of search agents (solutions) exchange information and combine them to come up with the best solution. This method also demonstrates the advantage of herd predators in the wild, which is the main idea of population-based algorithms.

The population-based approach can also be divided into several categories based on their source of inspiration, such as the evolutionary-based group in which the ideas are drawn from the laws governing natural evolution. The most well-known algorithm in this group is the genetic algorithm (GA) [11] (which mathematically mimics Darwinian's evolution) and differential evolution (DE) [12]. The other popular group is swarm-based, which simulates the natural collective behavior of a swarm of birds or flocks. The well-known algorithm that falls in this category is particle swarm optimization (PSO) [13] and ant colony optimization (ACO) [14]. The recent with outstanding performance one is the whale optimization algorithm (WOA) [15]. There are many more categories of the population-based approach, and we visualize these methods in Fig. 1.

Especially in the last two decades, hundreds of new algorithms have been proposed to the research community after the No Free Lunch theorem (NFL) was published. NFL expresses that no algorithm outperforms all other algorithms regarding all optimization problems [16]. If an algorithm X is better than algorithm Y in some optimization problems, then algorithm Y is superior to algorithm X in other optimization problems. Researchers and scholars have designed a wide range of MHAs based on this motivation. Although there are many proposed MHAs optimizer, there are not many MHA libraries that are packaged into an open-source code framework. Therefore, in this paper, we propose an open-source Python library specifically for MHAs optimizer called Mealpy. The library's primary goal is to provide a user-friendly and easy-to-use environment, whether for solving a simple test benchmark function or complex mathematical problems. As a result, it will save users from mathematics and programming burdens while solving optimization problems.

We then compare **Mealpy** with other general-purpose MHAs packages that used Python language, namely **Opytimizer**[1], **NiaPy**,[2] **EvoloPy**[3] [17], **DEAP**[4] [18], **Inspyred**[5], **MHA**[6], **yabox**,[7] **swarmlib**,[8] **FreeLunch**.[9]

[1] https://github.com/gugarosa/opytimizer
[2] https://github.com/NiaOrg/NiaPy
[3] https://github.com/7ossam81/EvoloPy
[4] https://github.com/DEAP/deap
[5] https://github.com/aarongarrett/inspyred
[6] https://github.com/tadatoshi/metaheuristic_algorithms_python
[7] https://github.com/pablormier/yabox
[8] https://github.com/HaaLeo/swarmlib
[9] https://github.com/MDCHAMP/FreeLunch

Table 1
Review of the main features of different Python-based packages/ libraries/ frameworks for MHAs.

Feature	Mealpy	Opytimizer	NiaPy	EvoloPy	DEAP	Inspyred	MHA	yabox	swarmlib	FreeLunch
User-friendly	yes	yes	yes	yes	no	yes	no	yes	no	no
Fully packaged library	yes	yes	yes	no	yes	yes	no	yes	yes	yes
Integrate with other frameworks	yes	yes	no	no	no	no	no	no	no	no
Documentation	yes	yes	yes	no	yes	no	no	no	no	no
Examples	yes	yes	yes	yes	yes	yes	no	yes	no	no
Testing	yes	yes	yes	no	no	yes	yes	yes	yes	yes
Support convergence chart	yes	yes	no	yes	no	no	no	yes	yes	no
Support other visualization	yes	no	no	no	no	no	no	yes	yes	no
Parallelization	yes	no	no	no	yes	no	no	yes	no	no
Focus field	All MHAs[a]	All MHAs	EC, Swarm	EC[b]	EC	EC	EC, Swarm	EC	Swarm[c]	EC, Swarm
Trending algorithms	yes	yes	no	no	no	no	no	no	no	no
Simplicity	yes	no	yes	no	no	no	no	yes	yes	no
# algorithms[d]	> 160	<= 100	<= 40	14	<= 10	<= 10	<= 4	2	7	6
PyPI	yes	yes	yes	no	yes	yes	yes	yes	yes	yes
Latest release	2023	2022	2022	N/A	2020	2015	2015	2020	2020	2023
Latest version	2.4.2	3.1.1	2.0.4	N/A	1.3.1	1.0.1	0.1.6	1.1.0	0.14.1	0.0.15
Total commits	1344	807	1423	184	2187	67	21	74	246	352
First commit	03/2020	11/2017	02/2018	05/2016	03/2010	03/2012	08/2015	07/2017	12/2018	11/2020
Last commit	03/2023	02/2023	12/2022	06/2022	01/2022	11/2021	12/2015	12/2022	12/2020	01/2023
Total downloads	106751	48642	114222	N/A	6130452	99758	55696	45541	65674	22660
Users per year	35583	8107	22844	N/A	471573	9068	7956	7590	13134	7553

[a]All MHAs: All different types of metaheuristic field.
[b]EC: Evolutionary-inspired Algorithms.
[c]Swarm: Swarm-based Algorithms.
[d]# algorithms: Number of available algorithms in the library.

To have an objective view when comparing Mealpy and other libraries, in the following paragraphs, we show essential features of our library which are unavailable in other libraries. In terms of fully packaged libraries, only EvoloPy and MAP are not fully packaged into libraries and are straight implementations of algorithms. Besides, the lack of documentation and examples makes it difficult for users to understand coding and developing new experiments.

To provide an objective comparison of Mealpy and other libraries, we show essential features of our library that are not available in other libraries in the following paragraphs. In terms of fully packaged libraries, only EvoloPy and MAP are not fully packaged and are straight algorithm implementations. Furthermore, the lack of documentation and examples makes it difficult for users to understand coding and developing new experiments.

Inspired is a complete, user-friendly library with good examples and testing, but the lack of documentation prevents users from expanding and developing. Furthermore, this library does not support illustrations, such as convergence charts, runtime charts, and so on. Even though it has been available since 2015, it only supports a few algorithms from the Evolutionary-based computing branch and lacks trending metaheuristics algorithms.

On PyPI, the SwarmLib library has a large number of downloads. However, the authors have stopped supporting it since 2020. Furthermore, this library is complicated and requires users to be familiar with command-line interfaces on Linux or Windows terminals. Despite being fully packaged, it lacks documentation, examples, and integration support with other machine learning libraries. Additionally, its algorithms are solely devoted to swarm-based metaheuristics.

Yabox was created and released before SwarmLib. It is an easy-to-use library that includes examples, testing, and graphics such as a convergence chart. However, it only has two algorithms: DE and PDE, with PDE being a parallel version of DE (can run on multiple process to speed up computation time). Notably, it lacks comprehensive documentation and only has a few notebook files to assist users in studying the available projects.

FreeLunch is a recently released, fully packaged library. However, unlike other libraries, it is not user-friendly, lacks documentation and examples, and does not support graph visualization. Furthermore, it only contains six algorithms from evolutionary and swarm-based MHAs, making it the least downloaded library on PyPI in this comparison.

NiaPy is a popular Python library with a large number of downloads on PyPi. It was first made available in 2018 and has been fully packaged since then. It includes approximately 40 algorithms, as well as complete documentation, examples, and testing to assist users. However, we could not find any documentation on graphic support in NiaPy, such as convergence charts. Although it has a large number of algorithms, they are all from the outdated evolutionary-based and swarm-based computing branches, and it lacks trendy MHAs like Mealpy. Furthermore, NiaPy does not support parallel computing for algorithms.

Another recent library that also implemented several MHAs is Opytimizer. Although it contains many of the same algorithms as Mealpy, it lacks visualization support such as the runtime chart, population diversity chart, and exploitation versus exploration chart. Furthermore, it lacks parallelization while Mealpy offers four operation types, including two parallel and two sequential modes for each algorithm. Additionally, the division of Opytimizer's package into too many small components makes comparing the implemented code and the source paper difficult, resulting in reduced transparency in the library.

Despite containing only a few Evolutionary-based computing algorithms, DEAP is one of the most widely used libraries. Although it does not support graphics, it remains popular due to its ability to perform parallelization. Only Mealpy and DEAP can run the algorithm in parallel mode. However, despite being released in 2010, DEAP still lacks the latest MHA algorithms and does not support integration with popular machine learning libraries such as TensorFlow and PyTorch. DEAP is more of a framework for building algorithms rather than a library of pre-built algorithms. Therefore, it is better suited for researchers in the computing field rather than beginners in meta-heuristics field.

Table 1 shows more detailed summary of the features of the different packages discussed here. There are aspects for comparison which we discuss as follows:

- Convergence: Does the package supports drawing a convergence chart of an algorithm that existed in it?
- Other visualization: Whether the package supports other visualization figures such as the processing time chart of an algorithm after generations, trajectory chart of search agents, the diversity of population chart, or the exploration verse exploitation chart of an algorithm?
- Fully packaged library: Is the package fully packaged into a library/ framework or not?

- Degree of meta-heuristic support: all kinds of meta-heuristic algorithms (All MHAs), or only focus on evolutionary-based computing algorithms (EC), or only possesses swarm-based algorithms (Swarm)
- Trending algorithms: Are recent meta-heuristic algorithms included?
- PyPI: Is it packaged and stored to PyPI for use with the pip management environment?
- Total downloads: We use third-party software PePy[10] to get the statistic count.

Contribution of the study. This paper proposes an open-source Python library for nature-inspired optimization algorithms named Mealpy. Python's OOP programming is utilized to generate less redundant and more readable code. Mealpy can solve an optimization problem as a black box. It is also compatible with many well-known machine learning and deep learning frameworks, including Scikit-Learn, Tensorflow, Keras, and Pytorch. Furthermore, each implemented algorithm in Mealpy has multiple learning modes (e.g., sequential, parallelization) that none of the other libraries has. To the best of our knowledge, Mealpy currently houses the most comprehensive collection of optimization algorithms with both traditional and cutting-edge meta-heuristics. We presented several examples and applications that can be solved with Mealpy. In addition, Mealpy has also been tested on CEC-2017 benchmark functions and real-world problems (hyper-parameter optimization of machine learning models).

Paper's structure. The remainder of the paper is organized as follows: Section 2 presents the our proposed library named Mealpy, its core components, guides how to use the proposed library and shows several working examples. Section 3 describes the experiments (benchmark function experiment and parallelization experiment), results, and discussion of our proposed library. Finally, Section 4 concludes the paper and provides ideas for future research.

2. Proposed library: Mealpy

In this section, we first present an overview of our proposed library named Mealpy. Then we described in more detail in core components of Mealpy. In general, Mealpy includes some main components as follows (the architecture of Mealpy is represented in Fig. 2):

- optimizer module: it contains a super-class Optimizer. The concept of inheritance in object-oriented programming (OOP) is utilized in this class. All algorithms will be a child of Optimizer. Therefore, all necessary methods in Optimizer will be inherited.
- utils package: it contains several helpful utility modules and classes such as Problem, Termination, Validator, Logger, History, and Visualization.
- the sub-packages classify meta-heuristic algorithms: Swarm-based package, Evolutionary-based package, Human-based package, Physics-based package, Bio-based package, System-based package, Math-based package, and Music-based package.

2.1. Optimizer module

The optimizer module contains the Optimizer class, which is the superclass of all algorithms in Mealpy. As it is known, most meta-heuristics algorithms have a standard structure. Therefore, it is logical to form a parent class, including properties and methods that child classes can inherit and use. Our proposed Optimizer class will include some essential properties of a meta-heuristics algorithm, such as the domain range (lower bound and upper bound) of the problem, visualization object, logging object, history object, the population of search agents and several helpful methods such as initialization(), evolve() and solve() function. In particular, the most important object is the population of search agents. All the methods and properties in this class work around the population of search agents. We describe in more detail how we designed the Population of search agents (a group of search agents) below. We also describe the Problem design, which describes a problem we need to solve.

2.1.1. Population of search agent design

Generally speaking, the majority of meta-heuristic algorithms are population-based algorithms. In which each algorithm starts from a random population. A population consists of a set of N elements called a search agent or a solution (a solution to the problem). The population then evolved in G life cycles (the maximum number of generations). The best search agent will be selected as the final solution to the problem. As can be seen, the design of how to store a population or a search agent of a meta-heuristic algorithm is vital. The speed and effectiveness of the algorithm can be affected by a bad design. Most of the latest libraries related to meta-heuristic algorithms used each search agent as a 1-dimensional vector and a population as a 2-dimensional matrix as Eqs. (1) and (2).

$$X_i = \begin{bmatrix} x_i^1 & x_i^2 & ... & x_i^D \end{bmatrix} \tag{1}$$

$$P = \begin{bmatrix} x_1^1 & x_1^2 & ... & x_1^D \\ x_2^1 & x_2^2 & ... & x_2^D \\ . & . & . & . \\ . & . & . & . \\ x_N^1 & x_N^2 & ... & x_N^D \end{bmatrix} \tag{2}$$

with N is the number of search agents (population size) and D is the number of dimensions (number of variables of the problem).

However, in Mealpy, we proposed a different structure of a population and search agent as shown in Fig. 3. From the user aspect, a search agent should know its position on the search space (a map of the search domain) and its fitness value (objective values that an optimizer uses to decide whether a search agent is good or not good (quantity aspect). Besides, depending on each algorithm's source of inspiration, such as bee, virus, or fish, each search agent can carry other properties. Therefore, we organize a solution as a Python list with required properties: *position* and *target_wrapper*. In which, *target_wrapper* is a Python list of single $fitness$ value and $list\_of\_objective$ value. For single-objective optimization, $list\_of\_objectives$ is a list of 1 real value. Meanwhile, for multi-objective optimization, $list\_of\_objectives$ is a list (or 1-D vector) of M objective values. The $fitness$ value is calculate as Eq. (3).

$$fitness = \sum_{i}^{M} w_i * obj_i; \tag{3}$$

where M is number of objective values, w_i is the weight corresponding to objective ith. The population is organized as a list of agents (Python list). The advantage of this design is that we can sort the population based on specific search agents' properties such as fitness or objective values. Moreover, the return population will be a list of sorted agents with their properties. For example: (sorted is a built-in Python function)

```
>>> ## sort and return population in ascending order of
    fitness value
>>> sorted_pop = sorted(population, key=lambda agent:
    agent[0])
>>>
>>> ## sort and return population in descending order
    of fitness value
>>> sorted_pop = sorted(population, key=lambda agent:
    agent[0], reverse=True)
>>>
>>> ## sort and return population in ascending order of
    the first objective value
>>> sorted_pop = sorted(population, key=lambda agent:
    agent[1][0])
>>>
```

[10] https://github.com/psincraian/pepy

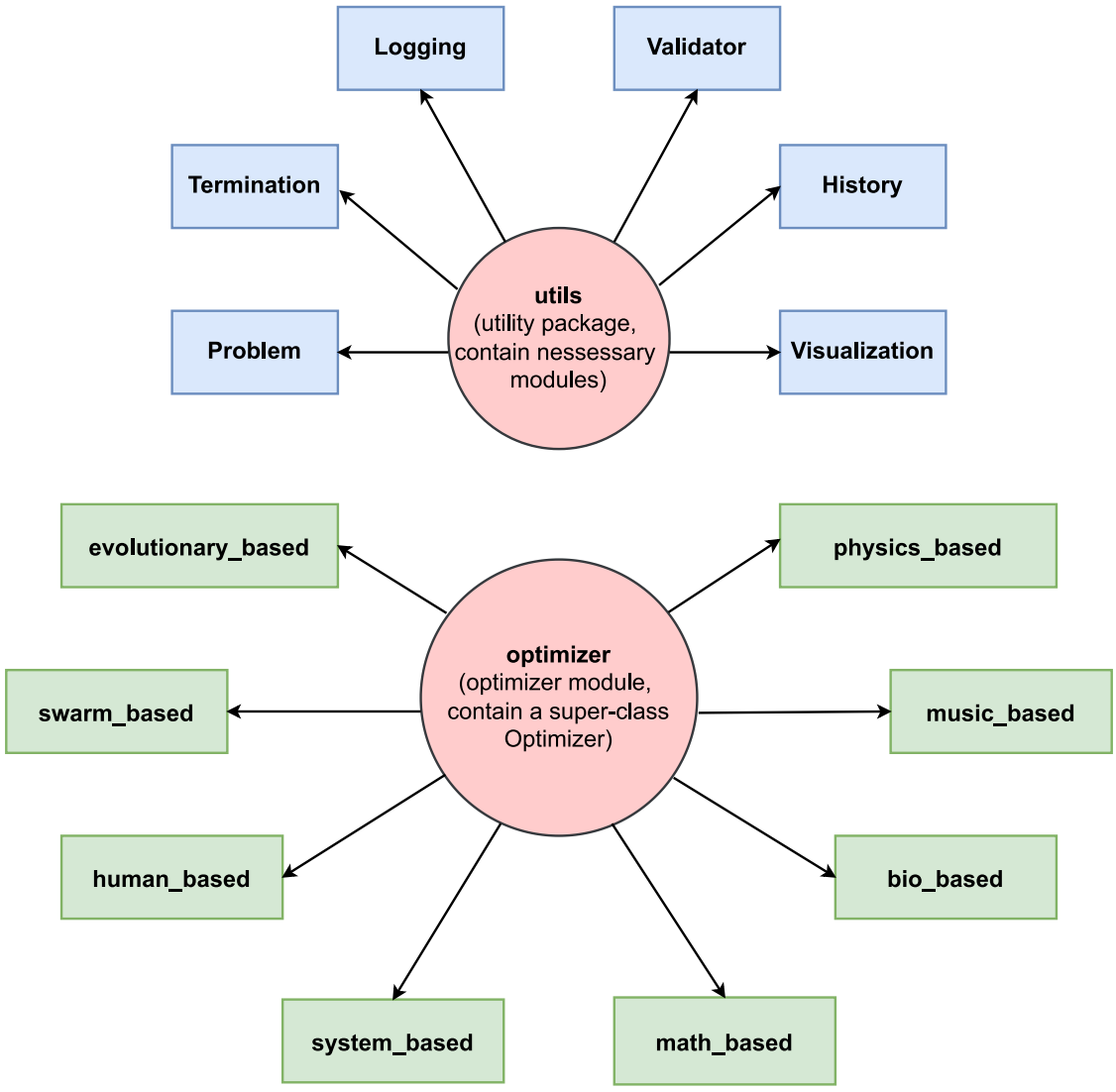


Fig. 2. Mealpy's components.

```
>>> ## sort and return population in ascending order of
      the second objective value
>>> sorted_pop = sorted(population, key=lambda agent:
     agent[1][1])
```

2.1.2. Problem design

A general mathematical form of an ongoing optimization problem is as follows:

```
minimize      f(x)
   x
subject to    g_i(x) <= 0, i = 1,..., m
              h_j(x) = 0,  j = 1,..., p
```

where $f : R^N -> R$: is the objective function to be minimized over the $N - variable$ vector x. $g_i(x) \leq 0$ are inequality constraints, $h_j(x) = 0$ are equality constraints and $m \geq 0$ and $p \geq 0$. If $m = p = 0$, then the problem is an unconstrained optimization problem. For the maximization problem, it can be treated by negating the objective function.

From the mathematical point of view, a problem composes the objective functions and search domain (the valid range for each variable in each dimension of the problem). The search domain should encode the lower and upper bounds for each decision variable expressed in the problem. Therefore, we must define a fitness function (objective functions) and a search domain (lower and upper bound of variables) for each problem. In Mealpy, we design the "problem" object as an input object (Python dictionary) for class Optimizer. So we can use Optimizer to solve different problems based on the input "problem" object. For example:

Listing 1: Define the sum square problem with 5 variables in range -100 to 100

```
>>> import numpy as np
>>>
>>> def fit_method(x):
>>>     return np.sum(x**2)
>>>
>>> problem_object = {
>>>     "fit_func": fit_method,
>>>     "lb": [-100, -100, -100, -100, -100],
>>>     "ub": [100, 100, 100, 100, 100],
>>>     "minmax": "min",
>>> }
```

2.2. Utils package

A utility package provides standard components shared across the framework, resulting in better reusability. This package provides the following modules:

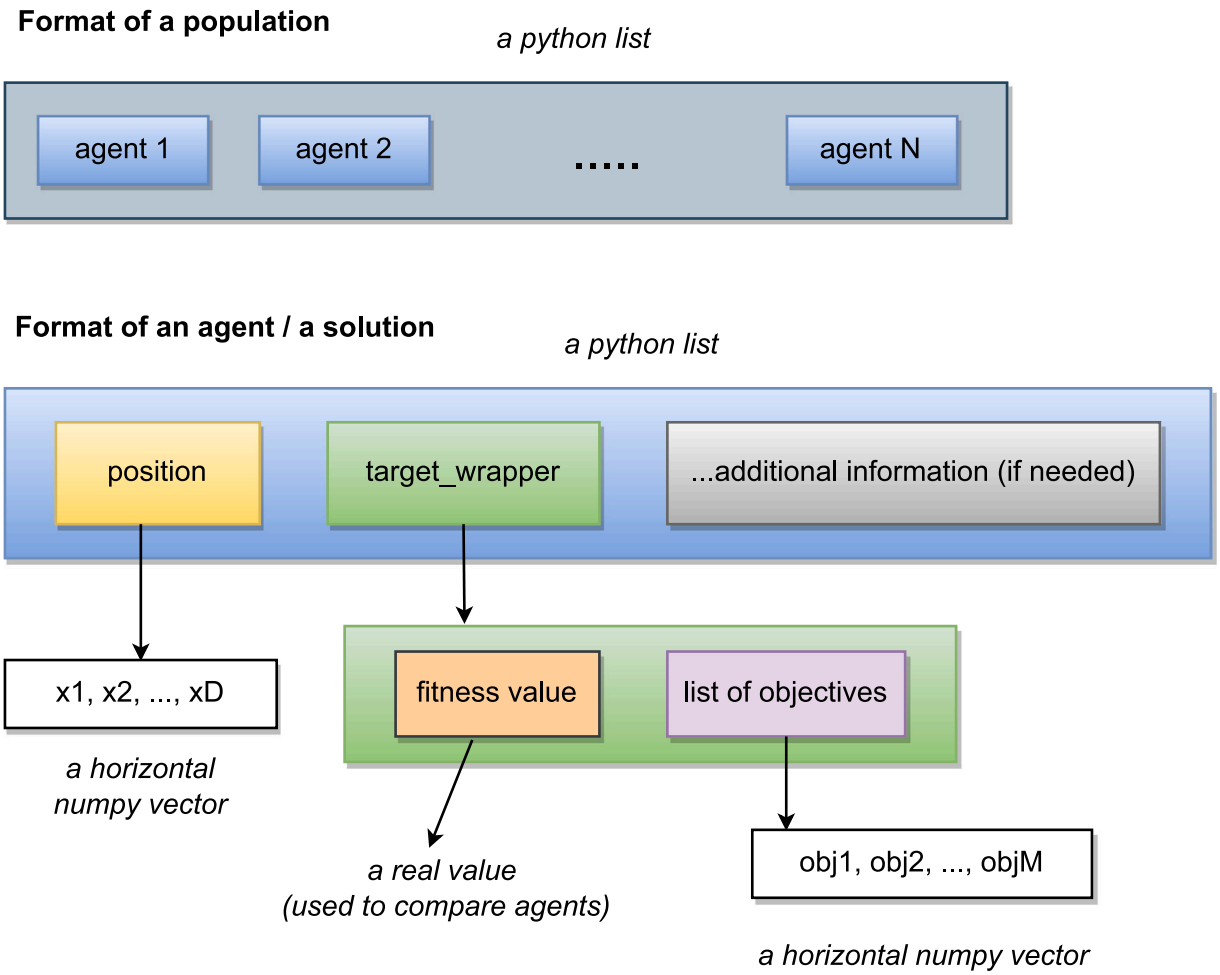


Fig. 3. Population and search agent structure.

- **Problem**: A class used to represent a problem in the mathematical form include some unique property such as fitness function, lower bound and upper bounds of variables, optimization problem (minimize or maximize), and a method to describe how to generate a solution for this particular problem, and how to modify the solution to the valid search range.
- **Termination**: A stopping condition forces the algorithm to terminate the search process. This class contains four termination types, including the number of generations, quality of solutions, time constraints, and the number of function evaluations. In addition, users can set multiple stopping conditions in combination with each other, which has many benefits. For example, increased reliability by reducing the likelihood of missing a critical stopping point, resulting in a more reliable and accurate solution. Improved efficiency since the problem-solving process can be optimized to end as soon as possible while ensuring a satisfactory solution.
- **Validator**: A module contains practical methods to verify and check the hyper-parameters of an algorithm with input constraints.
- **History**: After solving an optimization problem, a user may obtain additional information about search agents, such as fitness, objective values, and agent's history positions.
- **Visualization**: Provides every visual-related method, where the user can check a specific variable convergence (agent's history positions), the fitness function (objective functions) convergence, the evolving time of population through each epoch of a specific optimizer, the diversity of the population, and the exploration verse exploitation of a particular algorithm.
- **Logger**: Every method or action invoked in the library is logged into a log file. One can follow the log to find potential errors, essential warnings, or success messages throughout the evolving progress.

2.3. Optimizer and its sub-packages

Meta-heuristic algorithms are high-level search algorithms that incorporate multiple strategies to establish the exploration and exploitation procedure. From a developer aspect, the meta-heuristic algorithm is a class containing some search methods (local search, global search), operation methods (mutation, crossover, hunting, flying, ...), and a complex learning procedure that endeavors to find the most reasonable solution for a specific problem. Each meta-heuristic has its learning strategy and profoundly depends on the source of inspiration, such as hunting behaviors of a swarm of birds (PSO), fish (FFO), whale (WOA), and natural evolution theory (GA), among others. Algorithm 1 depicts a general pseudo-code of how a meta-heuristic algorithm works.

As mentioned above, due to the typical flow of most Meta-heuristic algorithm, using inheritance of object-oriented programming help to reduce code duplication by sharing standard features (properties and methods). In addition, it can save time and effort as the primary code need not be written again. Also, inheritance provides a clear model structure. Therefore, in Mealpy, a Meta-heuristic algorithm is implemented as a sub-class of the Optimizer class. We classify Meta-heuristic algorithms as different packages, each including a set of meta-heuristic algorithms. Currently, there are eight packages in Mealpy as follows:

Table 2
The implemented Meta-heuristic algorithms in evolutionary_based package (alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
Coral Reefs Optimization	Corals' biology and coral reefs formation	CRO	OriginalCRO [19] OCRO [20]	2014 2019
Differential Evolution	Iteratively improving a solution based on evolutionary process	DE	BaseDE [12] JADE [21] SADE [22] SHADE [23] L_SHADE [24] SAP_DE [25]	1997 2009 2005 2013 2014 2006
Evolution Strategies	Ideas of evolution in nature	ES	OriginalES [26] LevyES [27]	1971 2005
Evolutionary Programming	Simulated evolution as a learning process	EP	OriginalEP [28] LevyEP [29]	1964 2001
Flower Pollination Algorithm	The pollination process of flowering species	FPA	OriginalFPA [30]	2014
Genetic Algorithm	Charles Darwin's theory of natural evolution	GA	BaseGA [11] SingleGA[a] MultiGA[a] EliteSingleGA[a] EliteMultiGA[a]	1992 2022 2022 2022 2022
Memetic Algorithm	Extension version of GA with local search process	MA	OriginalMA [31]	1989

[a]Our developed version.

Algorithm 1 A general Meta-heuristic algorithm pseudo-code

Input: A mathematical form of the problem (fitness function, lower and upper bounds, minimize or maximize problem), the population size (ps), and the number of maximum generation (g_{max}).
Output: A global best solution (the best search agent)

```
Initialize a random population of N search agents.
Find the global best solution g_best
for g = 1 → g_max do
    for i = 1 → N do
        Perform the updating process of agent i − th
        Check the boundary of the new position
        Calculate the new fitness value
        Update the position of agent i − th if the new fitness is better than the old one
    Find the current global best solution c_best
    Update the global best solution g_best by c_best
Return: g_best
```

- evolutionary_based package: Contains algorithms based on natural evolutionary processes, such as Darwin's theory (GA), differential evolution (DE), ...Table 2 presents all implemented optimizers in this category.
- swarm_based package: Contains algorithms inspired by the behavior or social activities of the swarm of animals such as birds (PSO), whales (WOA)... Tables 3 and 4 shows all available optimizers in this category.
- human_based package: Includes algorithms inspired by human behavior and activities such as teaching learning-based optimizer (TLO), and brainstorm optimizer (BSO). Table 6 present all available algorithms in this category.
- physics_based package: Implemented all algorithms draw inspiration from physical processes present in nature such as multi-verse theory (MVO), Newton laws via tug of war game (TWO)... Table 5 shows all algorithms in this category.
- bio_based package: Algorithms mimic connectionism, social behavior, and microbiology's emergence to form biology models such as virus colony (VCS), slime mould (SMA)... Table 7 shows all implemented algorithms in this category.
- math_based package: Algorithms mimic the characteristics of the mathematical model, such as sin cosine function (SCA), arithmetic operators (AOA)... Table 9 presents all available algorithms in this category.
- system_based package: Contains algorithms that are motivated by the different types of system in nature, such as an artificial ecosystem (AEO), water cycle system (WCA)... Table 8 presents all employed optimizers in this category.
- Besides all the above packages, we provide a music_based package (taking ideas from a musical instrument or the improvisation of music players). Currently, it holds only a harmony search algorithm (HS).

2.4. Learning process

In this section, we will discuss the learning mechanisms of meta-heuristic algorithms in Mealpy. As can be observed from the general flow structure of the MHAs presented in Algorithm 1, we can calculate the complexity of a general algorithm as: $g_{max} * k * N * C$, where g_{max} represents the maximum number of generations, N is the number of search agents in the population (population size), k denotes the number of times the positions of the population are updated (which varies depending on the algorithm), and C is the cost of calculating the fitness function. However, based on this analysis, it is evident that the generations cannot be parallelized because the population of the next iteration is derived from the population of the previous iteration, which must be done in order. While we can perform position updates of multiple search agents simultaneously (possibly in parallel), this approach may affect several algorithms, as some algorithms update the current location of a search agent based on the value of its neighbors or the position of a random agent in the population.

In practice, the speed of an MHA optimizer depends mainly on the speed of the fitness function calculation. For example, for benchmarking function problems, the cost C of the fitness function can be completed within a few seconds. However, when it comes to finding the best set of parameters for a neural network model, the cost C of fitness can take several minutes or even hours if the network model is large and has many parameters to be tuned. Because the fitness function, in this case, requires the training and testing procedures of the network, followed by returning the performance metrics as fitness values. Therefore, one possible way to parallelize an MHA algorithm is to calculate the fitness function for multiple search agents concurrently using multiple CPUs or threads. Since calculating the fitness of multiple agents can run independently of each other and does not require any communication, this approach is feasible. Mealpy proposes four

Table 3
The implemented Meta-heuristic algorithms in swarm_based package (alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
African Vultures Optimization Algorithm	African vultures' lifestyle	AVOA	OriginalAVOA [32]	2022
Ant Colony Optimization	Simulate social behavior of grouping of ants in real environments	ACO-R	OriginalACOR [14]	2008
Ant Lion Optimizer	Mimics the hunting mechanism of antlions in nature	ALO	OriginalALO [33] BaseALO[a]	2015 2021
Aquila Optimizer	Natural behavior of the Aquila during the prey-hunting	AO	OriginalAO [34]	2021
Artificial Bee Colony	Mimics the process of honey bee swarms foraging for food.	ABC	OriginalABC [35]	2007
Artificial Gorilla Troops Optimization	Gorilla troops' social intelligence in nature	AGTO	OriginalAGTO [36]	2021
Artificial Rabbits Optimization	Survival strategies of rabbits in nature	ARO	OriginalARO [37]	2022
Bacterial Foraging Optimization	The foraging behavior of E. Coli bacteria	BFO	OriginalBFO [38] ABFO [39]	2002 2019
Bald Eagle Search	Fish hunting strategy and intelligent social behavior of bald eagles.	BES	OriginalBES [40]	2020
Bat Algorithm	Echolocation behavior of microbats	BA	OriginalBA [41] AdaptiveBA [42] ModifiedBA [43]	2010 2013 2015
Bees Algorithm	Mimics the natural food foraging behavior of honey bees.	BeesA	OriginalBeesA [44] ProbBeesA [45]	2005 2015
Bird Swarm Algorithm	The behavior of social iteration of birds in nature	BSA	OriginalBSA [46]	2016
Cat Swarm Optimization	The behavior of cats in the real world.	CSO	OriginalCSO [47]	2006
Coyote Optimization Algorithm	The social organization of the coyotes and its adaptation to the environment	COA	OriginalCOA [48]	2018
Cuckoo Search Algorithm	The obligate brood parasitism of some cuckoo species by laying their eggs in the nests of host birds of other species.	CSA	OriginalCSA [49]	2009
Dragonfly Optimization	The dragonflies' swarming behavior which is static and dynamic.	DO	OriginalDO [50]	2016
Dwarf Mongoose Optimization Algorithm	Inspired by the foraging and social behavior of dwarf mongoose.	DMOA	OriginalDMOA [51] DevDMOA[a]	2022 2022
Elephant Herding Optimization	The herding behavior of elephants in real world	EHO	OriginalEHO [52]	2015
Firefly Algorithm	The flashing behavior of fireflies.	FFA	OriginalFFA [53]	2009
Fireworks Algorithm	The simulation of fireworks explosion	FA	OriginalFA [54]	2010
Fruit-fly Optimization Algorithm	The foraging behavior of fruit fly	FOA	OriginalFOA [55] WhaleFOA [56] BaseFOA[a]	2012 2020 2021
Grasshopper Optimization Algorithm	The foraging and swarming behavior of grasshoppers in nature	GOA	OriginalGOA [57]	2017
Grey Wolf Optimizer	Simulates the leadership hierarchy and hunting mechanism of grey wolves in nature	GWO	OriginalGWO [58] RW_GWO [59] GWO_WOA [60]	2014 2019 2022

[a]Our developed version.

training methods for each MHA algorithm, as illustrated in Table 10. The flow of different learning strategies is also visualized in Fig. 4.

2.5. Mealpy's API

In this section, we describe how to use Mealpy. More details on concepts, strategies, and techniques can be found in documentation.[11] Mealpy's documentation is an ideal source to understand how the framework was built and to contribute to it in the future.

2.5.1. Installation

We design Mealpy as a one-to-go library that requires only one line of code to install. The following command can be run under any Python environment (raw, virtualenv, conda) regardless of any operating system such as Windows, Linux-based, or Mac OS.

```
\$ pip install mealpy
```

[11] https://mealpy.readthedocs.io/

After the installation, it can be imported from the Python environment:

```
>>> import mealpy
>>> print(mealpy.__version__)
2.5.0
```

2.5.2. Examples

The results in this paper were obtained using Python 3.8.5 with the Mealpy 2.5.0 package. Python can be found at https://www.python.org/, other packages used are available from https://pypi.org/. The example codes are publicized at https://github.com/thieu1995/mealpy_examples. Before embarking on details of some examples, we provide a general rule to work with Mealpy as follows to solve an optimization problem.

- Import essential libraries
- Define the fitness function
- Define the problem dictionary (a mathematical form of the problem)

Table 4
(Continue) The implemented Meta-heuristic algorithms in swarm_based package (alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
Harris Hawks Optimization	Mimic the hunting collaboration of Hawk's team.	HHO	OriginalHHO [61]	2019
Honey Badger Algorithm	The intelligent foraging of honey badger	HBA	OriginalHBA [62]	2022
Hunger Games Search	Mimics the concept of hunger-driven activities and behavioral choice of animals.	HGS	OriginalHGS [63]	2021
Jaya Algorithm	"JAYA" in Sanskrit means "victory". Combining the features of evolutionary algorithms and swarm-based intelligence.	JA	OriginalJA [64] BaseJA[a] LevyJA [65]	2016 2021 2021
Manta Ray Foraging Optimization	Assumes the actions of manta rays in catching the prey.	MRFO	OriginalMRFO [66]	2020
Marine Predators Algorithm	Simulating the rules that naturally govern in optimal foraging strategy and encounters rate policy between predator and prey in marine ecosystems	MPA	OriginalMPA [67]	2020
Moth Flame Optimization	The navigation method of moths in nature called transverse orientation.	MFO	OriginalMFO [68] BaseMFO[a]	2015 2021
Moth Search Algorithm	Inspired by the phototaxis and Lévy flights of the moths	MSA	OriginalMSA [69]	2018
Nake Mole-Rat Algorithm	Mimics the mating patterns of naked mole-rat present in nature.	NMRA	OriginalNMRA [70] ImprovedNMRA[a]	2019 2021
Particle Swarm Optimization	Inspired by the motion of bird flocks and schooling fish.	PSO	OriginalPSO [13] PPSO [71] HPSO_TVAC [72] C_PSO [73] CL_PSO [74]	1995 2019 2017 2015 2006
Pathfinder Algorithm	The collective movement of animal group and mimics the leadership hierarchy of swarms to find best food area or prey.	PFA	OriginalPFA [75]	2019
Sailfish Optimizer	Simulated the search, attack, hunting, and catching prey of the sailfish's group.	SFO	OriginalSFO [76] ImprovedSFO [77]	2019 2021
Salp Swarm Optimization	Simulating the swarm behavior of salps in nature.	SSO	OriginalSSO [78]	2017
Sand Cat Swarm Optimization	Mimics the sand cat behavior that tries to survive in nature	SCSO	OriginalSCSO [79]	2022
Sea Lion Optimization	Imitates the hunting behavior of sea lions in nature	SLO	OriginalSLO [80] ModifiedSLO [81] ImprovedSLO [82]	2019 2021 2022
Social Spider Algorithm	Imitating spiders' behaviors in nature	SSpiderA	OriginalSSpiderA [83]	2015
Social Spider Optimization	The cooperative characteristics of the social spider.	SSpiderO	OriginalSSpiderO [84]	2018
Sparrow Search Algorithm	Simulates the group wisdom foraging and anti-predation behaviors of sparrows.	SSA	OriginalSSA [85] BaseSSA[a]	2020 2021
Spotted Hyena Optimizer	Simulating the behavior of spotted hyenas.	SHO	OriginalSHO [86]	2017
Swarm Robotics Search And Rescue	The swarm artificial intelligence of coordinated robots	SRSR	OriginalSRSR [87]	2017
Tuna Swarm Optimization	The cooperative foraging behavior of tuna swarm.	TSO	OriginalTSO [88]	2021
Whale Optimization Algorithm	Mimics the hunting behavior of humpback whales.	WOA	OriginalWOA [15] HI_WOA [89]	2016 2019

[a]Our developed version.

- Call the model and set the hyper-parameters for the model
- Train the model and get the results
- Show the results

Note that there are two ways to call an optimizer from Mealpy as follows (In this study, we prefer to use the second way):

```
>>> ## from mealpy.package_name.module_name import
    class_name
>>> ## from mealpy.package_name import module_name
>>>
>>> from mealpy.swarm_based.PSO import BasePSO
>>> BasePSO(...)
>>>
>>> from mealpy.swarm_based import PSO
>>> PSO.BasePSO(...)
```

Example of solving a single unconstrained problem.

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{4}$$

For a simple illustration of the workflow above, the WOA algorithm from the swarm_based package is used to solve a simple problem: Finding the minimized value of sphere function with five dimensions and the domain range for all variables belonging to [−10, 10]. The mathematical form of the sphere function is shown in Eq. (4). The code that solves this problem can be operated as

Listing 2: Example code of solving a single unconstrained problem

```
>>> import numpy as np
>>> import mealpy.swarm_based import WOA
>>> ## Set the random seed to demonstrate the
    replicable experiment
>>> np.random.seed(12345)
>>>
>>> def fit_method(x):
>>>    return np.sum(x**2)
>>>
>>> problem_object = {
>>>    "fit_func": fit_method,
>>>    "lb": [-100, -100, -100, -100, -100],
>>>    "ub": [100, 100, 100, 100, 100],
>>>    "minmax": "min",
>>> }
```

Table 5
The implemented Meta-heuristic algorithms in physics_based package (alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
Archimedes Optimization Algorithm	Inspired by the law of physics Archimedes' Principle.	ArchOA	OriginalArchOA [90]	2021
Atom Search Optimization	Mimics the atomic motion model in nature.	ASO	OriginalASO [91]	2019
Electromagnetic Field Optimization	The behavior of electromagnets with different polarities and the golden ratio	EFO	OriginalEFO [92] BaseEFO[a]	2016 2021
Equilibrium Optimizer	Derived from control volume mass balance models used to estimate both dynamic and equilibrium states.	EO	OriginalEO [93] ModifiedEO [94] AdaptiveEO [95]	2019 2020 2020
Henry Gas Solubility Optimization	Mimics the behavior governed by Henry's law.	HGSO	OriginalHGSO [96]	2019
Multi-Verse Optimizer	Based on the concepts of the multiverse theory.	MVO	OriginalMVO [97] BaseMVO[a]	2016 2021
Nuclear Reaction Optimization	Imitates the nuclear reaction process and consists of two phases, namely, nuclear fission (NFi) and nuclear fusion (NFu) phase.	NRO	OriginalNRO [98]	2019
Simulated Annealing	Models the physical process of heating a material and then slowly lowering the temperature to decrease defects, thus minimizing the system energy.	SA	OriginalSA [10]	1987
Tug of War Optimization	Inspired by the game of tug of war of several opponent teams.	TWO	OriginalTWO [99] OppoTWO [100] LevyTWO[a] EnhancedTWO [101]	2016 2021 2021 2020
Wind Driven Optimization	Simulating the atmospheric motion.	WDO	OriginalWDO [102]	2013

[a]Our developed version.

Table 6
The implemented Meta-heuristic algorithms in human_based package(alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
Battle Royale Optimization	Inspired by a genre of digital games knowns as "battle royale."	BRO	OriginalBRO [103] BaseBRO[a]	2020 2021
Brain Storm Optimization	Inspired by the human brainstorming process.	BSO	OriginalBSO [104] ImprovedBSO [105]	2011 2017
Coronavirus Herd Immunity Optimization	Imitates the herd immunity strategy to eliminate of the COVID-19 disease.	CHIO	OriginalCHIO [106] BaseCHIO[a]	2020 2021
Culture Algorithm	Inspired by the conceptual models of the human cultural evolution process.	CA	OriginalCA [107]	1994
Forensic-Based Investigation Optimization	Inspired by the suspect investigation–location–pursuit process that is used by police officers.	FBIO	OriginalFBIO [108] BaseFBIO [109]	2020 2021
Gaining Sharing Knowledge-based Algorithm	Mimics the process of gaining and sharing knowledge during the human life span.	GSKA	OriginalGSKA [110] BaseGSKA [111]	2019 2020
Imperialist Competitive Algorithm	Inspired by the imperialistic competition.	ICA	OriginalICA [112]	2007
Life Choice-Based Optimization	Typical decision-making ability of humans to attain their goals while learning from fellow members.	LCO	OriginalLCO [113] BaseLCO[a] ImprovedLCO[a]	2019 2021 2021
Queuing Search Algorithm	Inspired from human activities in queuing	QSA	OriginalQSA [114] BaseQSA[a] OppoQSA [115] LevyQSA [116] ImprovedQSA [117]	2019 2021 2022 2021 2021
Search And Rescue Optimization	Imitates the explorations behavior of humans during search and rescue operations.	SARO	OriginalSARO [118] BaseSARO[a]	2019 2021
Social Ski-Driver Optimization	Simulate the process of ski-driving of groups of people.	SSDO	OriginalSSDO [119]	2019
Student Psychology Based Optimization	Based on the psychology of students attempting to improve their examination performance to become the best student in the class.	SPBO	OriginalSPBO [120] DevSPBO[a]	2020 2022
Teaching Learning-based Optimization	Based on the effect of the influence of a teacher on the output of learners in a class.	TLO	OriginalTLO [121] BaseTLO [122] ITLO [123]	2011 2012 2012

[a]Our developed version.

```
>>>
>>> algorithm = WHO.BaseWOA(epoch=100, pop_size=50)
>>> final_postition, final_fitness = algorithm.solve(
    problem_object)
>>> print(f"Best solution: {final_position}, fitness: {
    final_fitness}")
>>>
```

```
>>> ## To get all of available figures
>>> algorithm.history.save_global_best_fitness_chart()
>>> algorithm.history.save_local_best_fitness_chart()
>>> algorithm.history.save_diversity_chart()
>>> algorithm.history.save_global_objectives_chart()
>>> algorithm.history.save_local_objectives_chart()
```

Table 7
The implemented Meta-heuristic algorithms in bio_based package (alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
Barnacles Mating Optimizer	Mimics the mating behavior of barnacles in nature.	BMO	OriginalBMO [124]	2020
Biogeography-Based Optimization	Inspired by the biogeography concept (i.e., the idea of a species migration strategy).	BBO	OriginalBBO [125] BaseBBO[a]	2008 2021
Earthworm Optimization Algorithm	Inspired by the earthworm's contribution to nature (aeration of the soil through burrowing and enrichment of the soil through waste nutrients),	EOA	OriginalEOA [126]	2018
Invasive Weed Optimization	Inspired from the invasive behavior of weeds growth in nature.	IWO	OriginalIWO [127]	2006
Satin Bowerbird Optimizer	Simulate the reproduction and migration of Bowerbirds in the wild.	SBO	OriginalSBO [128] BaseSBO[a]	2017 2021
Seagull Optimization Algorithm	Imitates the seagull's foraging behavior, and migration and attack behavior are the seagull's foraging ways.	SOA	OriginalSOA [129] DevSOA[a]	2019 2022
Slime Mould Algorithm	Based on the oscillation style of slime mould in nature.	SMA	OriginalSMA [130] BaseSMA[a]	2020 2021
Symbiotic Organisms Search	Simulates the symbiotic interaction strategies adopted by organisms to survive and propagate in the ecosystem.	SOS	OriginalSOS [131]	2014
Tunicate Swarm Algorithm	Inspired by small sea creatures, the tunicates, and their swarm behavior.	TSA	OriginalTSA [132]	2020
Virus Colony Search	Simulates virus diffusion and infection strategies used by host cells to survive and propagate in the cell environment.	VCS	OriginalVCS [133] BaseVCS[a]	2016 2021
Wildebeest Herd Optimization	Inspired by Wildebeest herding behavior during reproduction and social interaction.	WHO	OriginalWHO [134]	2019

[a]Our developed version.

Table 8
The implemented Meta-heuristic algorithms in system_based package (alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
Germinal Center Optimization	Hybridization between evolutionary computing and Artificial Immune System based on the germinal center reaction.	GCO	OriginalGCO [135] BaseGCO[a]	2018 2020
Water Cycle Algorithm	Inspired from nature and based on water cycle process in a nature system	WCA	OriginalWCA [136]	2012
Artificial Ecosystem-based Optimization	Inspired by the flow of energy in an ecosystem on the earth, this algorithm mimics three unique behaviors of living organisms, including production, consumption, and decomposition.	AEO	OriginalAEO [137] EnhancedAEO [138] ModifiedAEO [139] ImprovedAEO [140] AugmentedAEO [141]	2019 2020 2020 2021 2023

[a]Our developed version.

```
>>> algorithm.history.
    save_exploration_exploitation_chart()
>>> algorithm.history.save_runtime_chart()
>>> algorithm.history.save_trajectory_chart()

Best solution: [-5.86140545e-11 -1.14845541e-10
    8.34714479e-11 -1.36054557e-10 -2.52322766e-11],
    fitness: 4.274009859170477e-20
```

In this example, we use the default learning mode "single". Depending on the specific problem, the user can choose different learning modes with the parameter "mode" in the "solve()" function. For example, we select multiple cores to solve this problem. The syntax can be as follows:

```
>>> algorithm.solve(problem_object, mode='process')
```

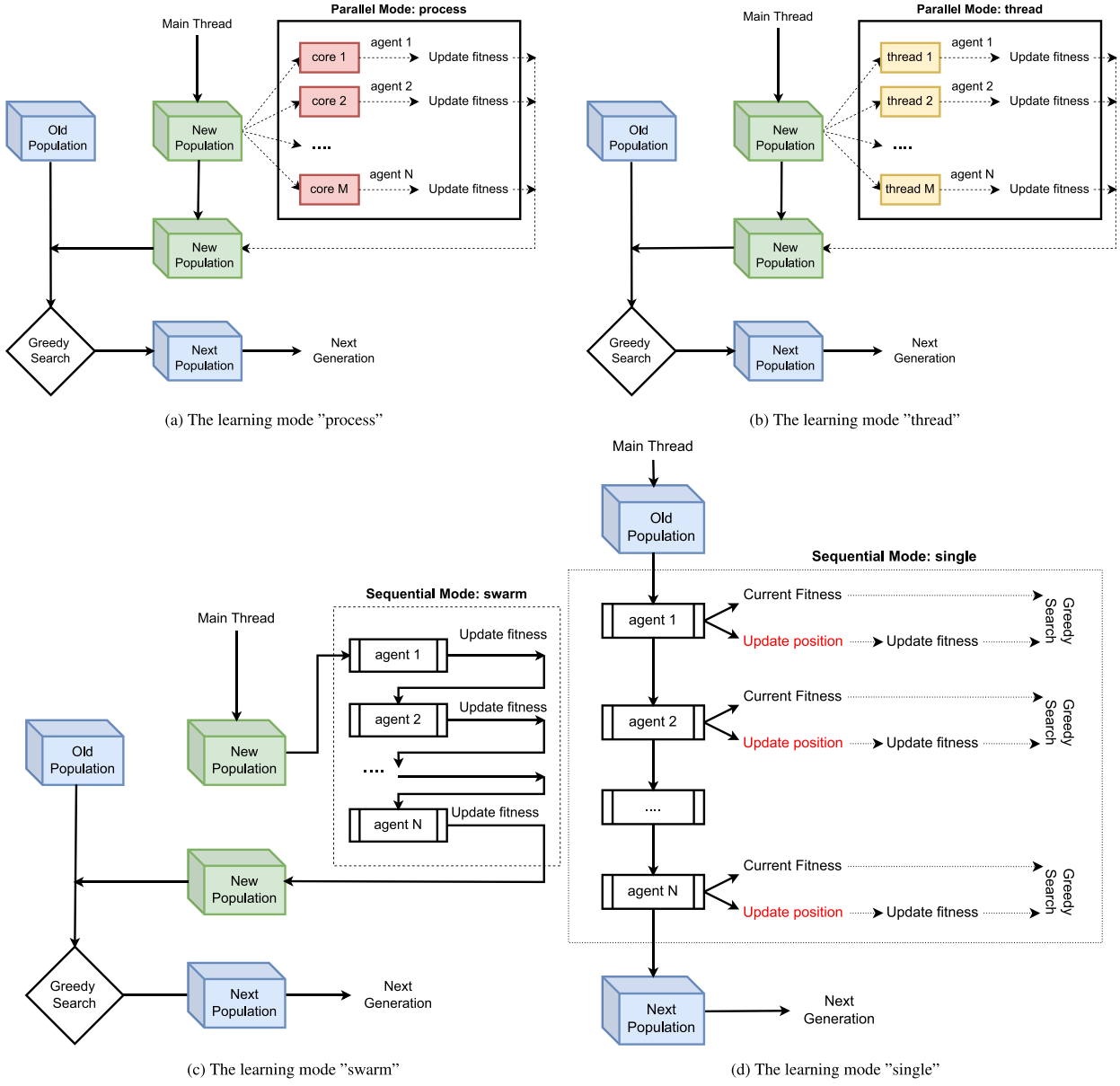
Or multiple threads to solve the problem as follows:

```
>>> algorithm.solve(problem_object, mode='thread')
```

We also proposed an embedded visualization module in Mealpy to make it more usable. All necessary visualized-related functions are integrated into the "history" object. It also makes sense since all data used in visualized functions are derived from historical objects. For example, the sphere function problem above is solved by the traditional WOA algorithm. Fig. 5 shows all currently available figures for the results.

- The global best fitness and local best fitness chart are derived from global best fitness, and local best fitness values found after generations. The global best fitness value is the best fitness value found so far (in all passed iterations), and the current best fitness value is the best fitness value found in the current iteration.
- The global objectives and local objectives chart are derived from the objectives of the global best solution and local best solution. The global best solution is the solution with the global best fitness value found so far (in all passed iterations), and the current best solution is the solution with the best fitness value found in the current iteration.
- The population's diversity chart and explore verse exploit percentages chart are derived from [152].
- The runtime chart describes the time to finish an iteration/generation/epoch in seconds
- The trajectory chart represents the movement of selected agents in selected dimensions.

Example of solving a single constrained problem. For dealing with the inequality constraints optimization problem, we recommend using the



(a) The learning mode "process"

(b) The learning mode "thread"

(c) The learning mode "swarm"

(d) The learning mode "single"

Fig. 4. Four learning modes in Mealpy.

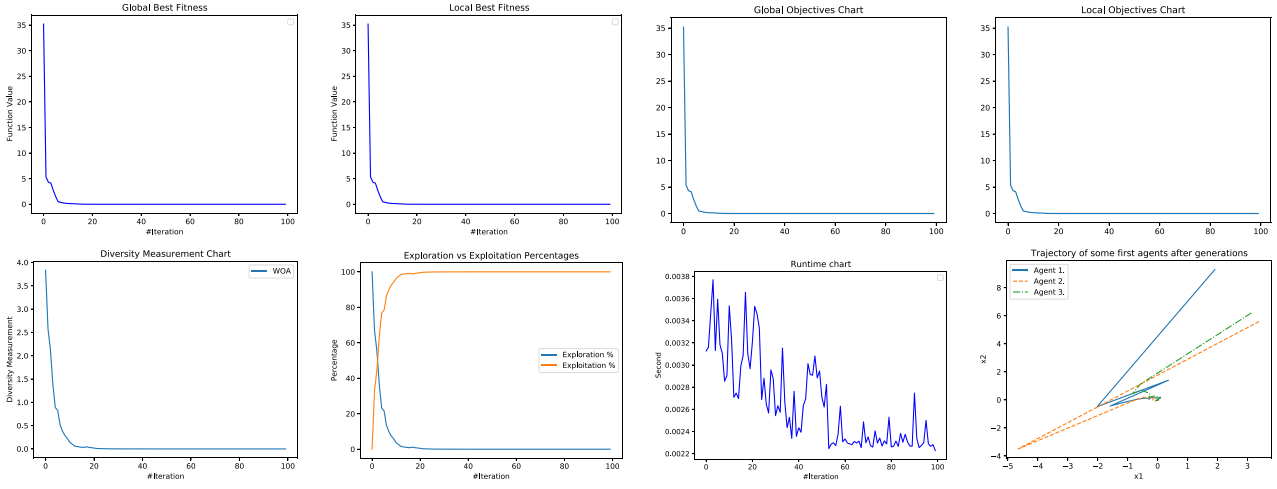


Fig. 5. The visualization results of the WOA algorithm on the sphere function problem.

Table 9
The implemented Meta-heuristic algorithms in math_based package (alphabetical order).

Algorithm name	Main idea	Module	Class name	Year
Arithmetic Optimization Algorithm	Utilizes the distribution behavior of the main arithmetic operators in mathematics.	AOA	OrginalAOA [142]	2021
Chaos Game Optimization	Based on some principles of chaos theory in which the configuration of fractals by chaos game concept and the fractals self-similarity issues are in perspective.	CGO	OriginalCGO [143]	2021
Circle Search Algorithm	Inspired by the geometrical features of circles such as diameter, center, perimeter, and tangent lines.	CircleSA	OriginalCircleSA [144]	2022
Cross-Entropy Method	a Monte Carlo method for importance sampling and optimization.	CEM	OriginalCEM [145]	1997
Gradient-Based Optimizer	Inspired by the gradient-based Newton's method.	GBO	OriginalGBO [146]	2020
Hill Climbing	Simulates the movement's direction of increasing elevation/value to find the mountain's peak.	HC	OriginalHC [147] SwarmHC[a]	1993 2021
Pareto-like Sequential Sampling	Inspired by Pareto's principle.	PSS	OriginalPSS [148]	2021
RUNge Kutta optimizer	Based on the mathematical foundations and ideas of the Runge Kutta (RK) method.	RUN	OriginalRUN [149]	2021
Sine Cosine Algorithm	Motivated by the trigonometric sine and cosine functions.	SCA	OriginalSCA [150] BaseSCA[a]	2016
weIghted meaN oF vectOrs	INFO is a weighted mean method with three core procedures: updating rule, vector combining, and a local search.	INFO	OriginalINFO [151]	2022

[a]Our developed version.

Table 10
Overview of the four training modes in Mealpy.

Type	Mode	Description
Parallel	process	The position of the new agent is stored in a separate Python list. Once the position updates for the entire population are complete, the new fitness for the updated position is calculated. This is done by multiple processes using the 'concurrent.futures' module. After this, the greedy strategy is utilized to select better solutions for both the old and new populations. It should be noted that the entire population is moved together.
	thread	The flow of this mode is similar to the 'process' mode. However, during the parallelization stage, fitness calculation for the population is carried out by multiple threads generated by a single process. It is important to note that the entire population is moved together.
Sequential	swarm	The new position of the agent is stored in a separate Python list. Once the update of positions for the entire population is complete, the new fitness for each new position is calculated. However, in this mode, fitness calculation is done for each agent in order. The whole population is moved in a specific order, and the updating process of the next agent is not affected by the previous agents.
	single	This is the default mode in Mealpy, and it is also a traditional mode that is commonly used in MHAs to run a model. The new position and corresponding fitness value are calculated and then compared with the old agent's position and fitness value. If the new fitness value is superior, it replaces the old agent. It is important to note that the whole population is moved in a specific order, and the updating process of posterior agents is affected by the preceding agents.

penalty method [153]. For example, the penalty function (Eq. (5)) handles the constraints' violation. We demonstrate this example using the traditional DE algorithm to solve the Rosenbrock function constrained with a cubic and a line [154]. The function has the global minimum at $x = y = 1.0$ and $f(1.0, 1.0) = 0$, the mathematical form is described as follows:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

$$s.t: (x-1)^3 - y + 1 \leq 0; \quad x + y - 2 \leq 0; \quad -1.5 \leq x \leq 1.5; \quad -0.5 \leq y \leq 2.5$$

$$penalty(z) = \begin{cases} 0, & \text{if } x \leq 0 \\ z, & \text{else if } 0 < z \leq 1 \\ z^2, & \text{otherwise} \end{cases} \tag{5}$$

Listing 3: Example code of solving a single constrained problem

```
>>> import numpy as np
>>> from mealpy.evolutionary_based import DE
>>> np.random.seed(12345)
>>>
>>> def penalty(value):
>>>     if 0 < value < 1:
>>>         return value
>>>     elif value >= 1:
>>>         return value**2
>>>     else:
>>>         return 0
>>>
>>> def fit_method(pos):
>>>     def fxy(pos):
>>>         return (1-pos[0])**2 + 100*(pos[1] - pos
    [0]**2)**2
>>>     def g1(pos):
>>>         return (pos[0]-1)**3 - pos[1] + 1
>>>     def g2(pos):
>>>         return pos[0] + pos[1] - 2
>>>     fitness = fxy(pos) + penalty(g1(pos)) + penalty
    (g2(pos))
>>>     return fitness
>>>
>>> problem_object = {
>>>     "fit_func": fit_method,
>>>     "lb": [-1.5, -0.5],
>>>     "ub": [1.5, 2.5],
>>>     "minmax": "min",
>>> }
```

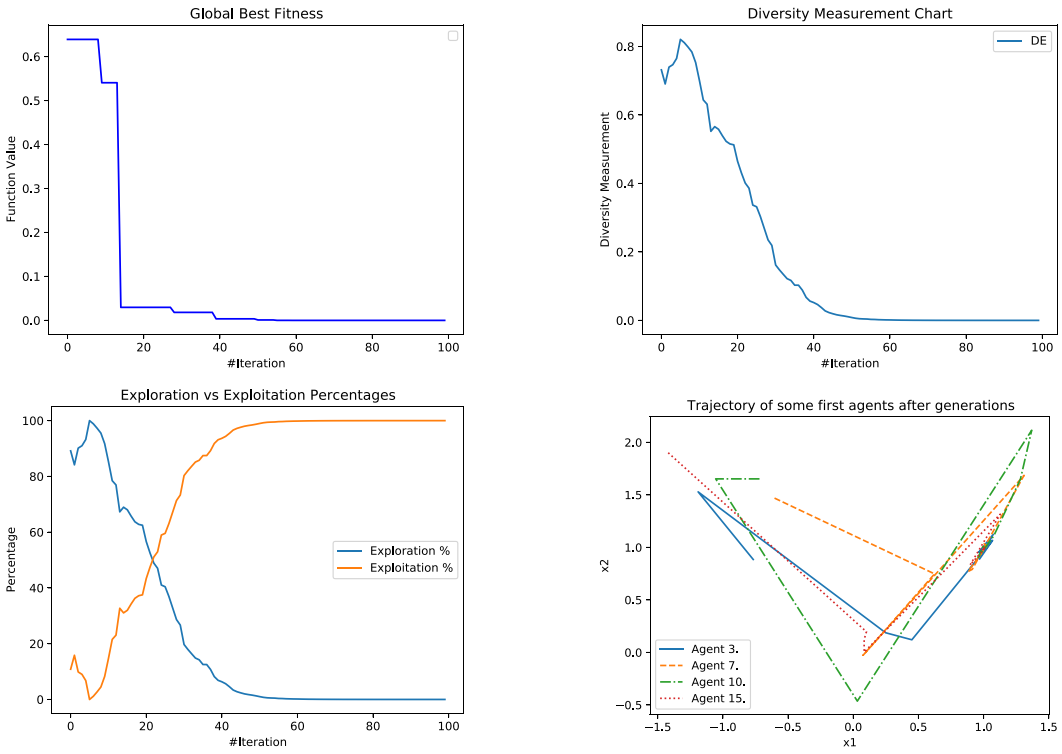


Fig. 6. The visualization results of the DE algorithm on constrained optimization problem.

```
>>>
>>> algorithm = DE.BaseDE(epoch=100, pop_size=50)
>>> final_solution, final_fitness = algorithm.solve(
    problem_object)
>>> print(f"Best solution: {final_solution}, fitness: {
    final_fitness}")
>>>
>>> algorithm.history.save_global_best_fitness_chart()
>>> algorithm.history.
    save_exploration_exploitation_chart()
>>> algorithm.history.save_diversity_chart()
>>> algorithm.history.save_trajectory_chart(
    list_agent_idx=[3,7,10,15])

Best solution: [0.99999714 1.00000002], fitness:
    3.3002708456738707e-09
```

As can be seen, the DE algorithm can easily reach the near-global best optimal point after 100 generations. We show some figure results of DE on constrained function in Fig. 6.

Example of solving multi-objective optimization problem. To demonstrate how to use Mealpy to solve multi-objective optimization, we use the TLO algorithm from the human-inspired group to solve Kursawe function [155]. The tested function has three variables in the range [−5, 5], and we need to find the minimum value satisfying both functions in Eq. (6).

$$minimize = \begin{cases} f_1(x) = \sum_{i=1}^{2}\left[-10 * exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2}\right)\right] \\ f_2(x) = \sum_{i=1}^{3}\left[|x_i|^{0.8} + 5sin(x_i^3)\right] \end{cases} \quad (6)$$

Listing 4: Example code of solving multi-objective optimization problem

```
>>> import numpy as np
>>> from mealpy.human_based import TLO
>>> np.random.seed(12345)
>>>
>>> def fit_method(pos):
>>>     def f1(pos):
>>>         return -10*(np.exp(-0.2*np.sqrt(pos[0]**2+
    pos[1]**2)) + \
>>>                 np.exp(-0.2*np.sqrt(pos[1]**2+pos
    [2]**2)))
>>>     def f2(pos):
>>>         return np.sum([np.abs(x)**0.8 + 5*np.sin(x
    **3) for x in pos])
>>>     obj1, obj2 = f1(pos), f2(pos)
>>>     return [obj1, obj2]
>>>
>>> problem_object = {
>>>     "fit_func": fit_method,
>>>     "lb": [-1.5, -0.5],
>>>     "ub": [1.5, 2.5],
>>>     "minmax": "min",
>>>     "obj_weights": [0.5, 0.5]
>>> }
>>>
>>> algorithm = TLO.BaseTLO(epoch=100, pop_size=50)
>>> algorithm.solve(problem_object)
>>> print(f"Best solution: {algorithm.solution[0]},\
    nTarget: {algorithm.solution[1]}")
>>>
>>> ## Get visualization figures
>>> algorithm.history.save_global_objectives_chart()
>>> algorithm.history.save_global_best_fitness_chart()
>>> algorithm.history.save_runtime_chart()
>>> algorithm.history.save_trajectory_chart(
    list_agent_idx=[6,13,24])

Best solution: [-1.13896027 -1.12436903 -1.13896028],
Target: [-13.052395491349895, [-14.521677344495188,
    -11.583113638204601]]
```

The difference between single and multi-objective optimization is the number of functions to be optimized. While single optimization attempts to optimize one objective function, multi-objective optimization can operate more than one objective function. In mealpy, we provide a simple but efficient weighted-sum method to solve the multi-objective optimization problem. This method combines multiple objective functions using weights to form a final objective function (or fitness function). The fitness value is calculated as in Eq. (3). Therefore, to solve multi-objective optimization, the user needs to set the parameter "obj_weights" in the "problem" object to determine the weight for

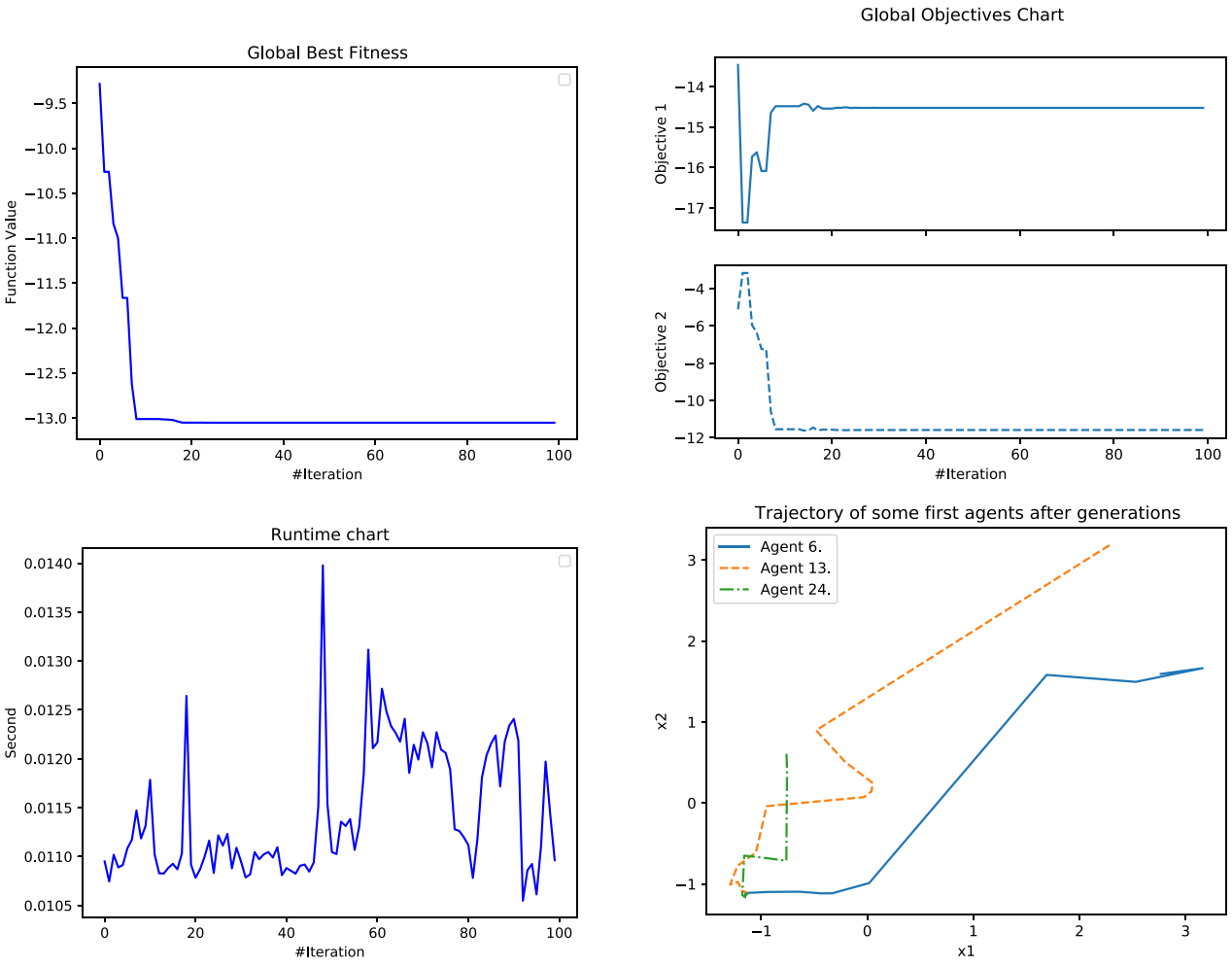


Fig. 7. The visualization results of the TLO algorithm on the multi-objective optimization problem.

each objective. The printed results showed that for this problem, the TLO algorithm gets the optimal fitness value around −13.05, the first objective value is −14.52, and the second objective value is −11.58. The result figures of the TLO algorithm for this particular problem are represented in Fig. 7.

More complicated examples. We provide many more complex scenarios and examples in the examples folder[12]:

- Use Mealpy to solve many problems at the same time (the script runs sequentially and in parallel to speed up the processing)
- Use Mealpy to solve many problems with many tries (scenario runs sequentially and in parallel to speed up the processing)
- How to use low-level package visualization in Mealpy without going through the history object.

2.5.3. Mealpy's applications

In this section, we describe how to run an application with Mealpy. Several pre-loaded applications are included with the library, and many integrations with well-known frameworks/libraries, such as Tensorflow, Keras, PyTorch, and Scikit-Learn.

Currently, the most common utilization of meta-heuristics algorithms is used to test various benchmark functions. Benchmark functions are the theoretical test for algorithms. We provide three basic examples to show the main features, which are single-objective optimization (run_single_objective_function.py), multi-objective optimization (run_multi_objective_functions.py), and constrained optimization (run_constraint_functions.py). It is located under the folder "examples/applications/" in our GitHub repository.

Besides, we provide many examples of using Mealpy to solve real-world problems. These examples include many well-known machine-learning frameworks used by the community. Each sub-folder in the above examples folder includes a variety of different instances, for example:

- **Keras (Tensorflow)**: Mealpy can replace gradient descent in neural network models to solve classification and regression problems. Alternatively, Mealpy can find the best hyper-parameters for different neural network models.
- **Pytorch**: Mealpy can find the best set of parameters for the complete regression of the SVM model
- **Scikit-learn**: Mealpy can find the best set of parameters for the classification problem of the SVM model. Note that models need to choose the best set of parameters, and the parameter set is in the form of real value, which conventional techniques like GridSearch or brute force search cannot do. However, Mealpy can solve this problem.

In addition, a discrete domain problem is one type of problem for which a traditional meta-heuristic algorithm is rarely used to find solutions. The meta-heuristic algorithm, as it is known, is an algorithm for continuous domain problems. Mealpy's meta-heuristics algorithms, on the other hand, can solve problems in the discrete domain.

[12] https://github.com/thieu1995/mealpy/tree/master/examples

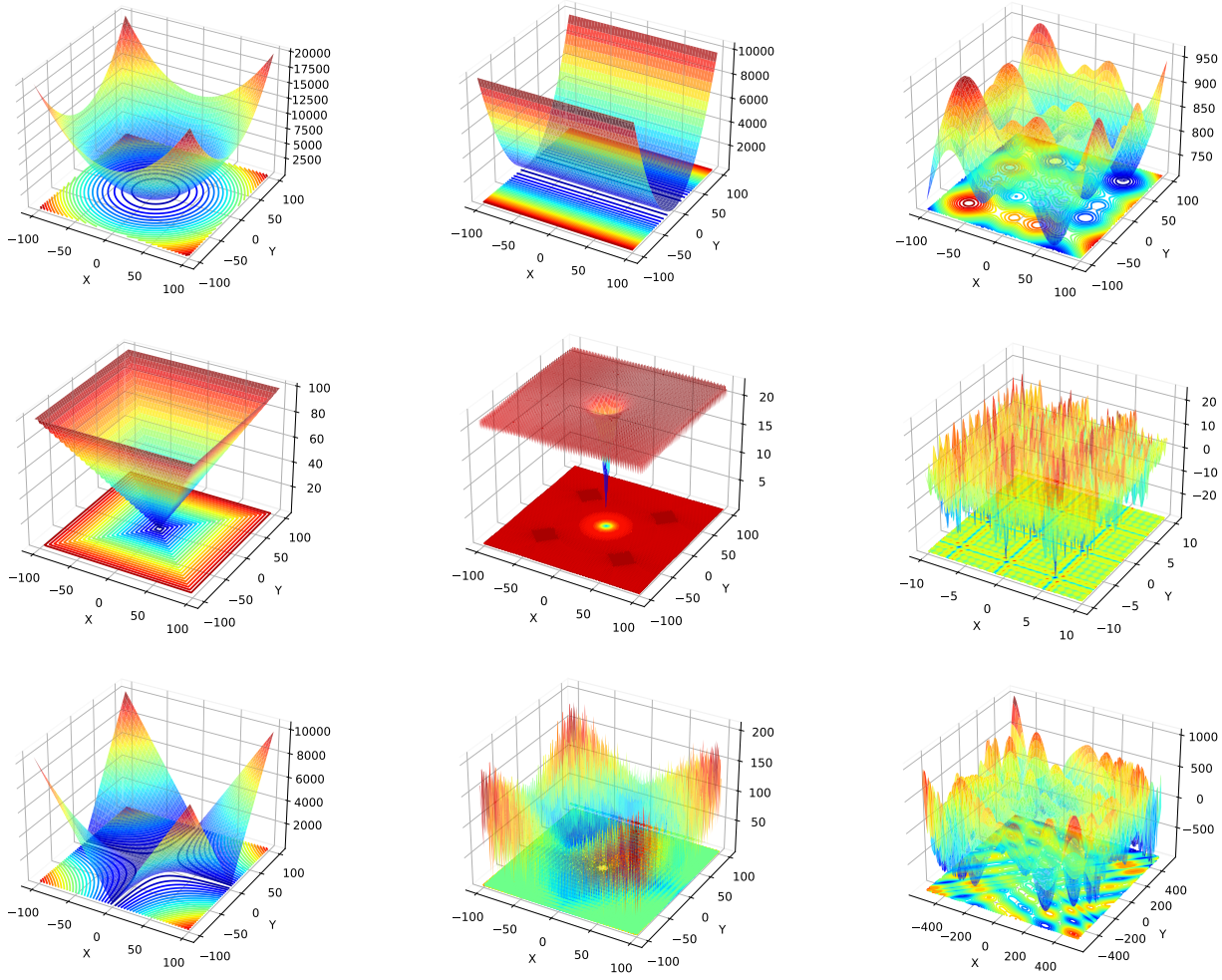


Fig. 8. Examples of 3D plot for some of benchmark functions.

The knapsack problem is the most well-known example. The folder "examples/discrete-problems/knapsack-problem.py" contains this example. Furthermore, we created two dedicated repositories on GitHub to show how to use Mealpy to solve traveling salesman problem[13] and feature selection problem[14].

3. Experiments and analysis

3.1. Numerical results of benchmark functions

A benchmark function, also known as a test function, is a mathematical function used to evaluate the performance of optimization algorithms. These functions are typically designed to be difficult to solve, meaning that finding the optimal solution is challenging and requires significant computational effort. Benchmark functions are used to compare the performance of different optimization algorithms and to assess their strengths and weaknesses. They provide a standardized way of testing the performance of algorithms, allowing researchers to compare results across different studies and to identify which algorithms perform best for different types of problems.

In this section, we demonstrate the ability of Mealpy's algorithms by solving 20 benchmark functions in the CEC-2017 competition (F1 to F20). They were designed for the IEEE Congress on Evolutionary Computation (CEC) 2017 competition. The function formula and characteristics details are in [156]. The 3D plots of several benchmark functions are presented in Fig. 8. We select 28 algorithms that belong to all eight categories of meta-heuristics, including:

- Evolutionary-based: DE, CRO, GA
- Math-based: CGO, GBO, PSS
- Bio-based: SMA, EOA, BBO
- System-based: AEO, GCO
- Human-based: TLO, FBIO, GSKA, QSA
- Music-based: HS
- Physics-based: HGSO, NRO, EFO, EO
- Swarm-based: DO, GWO, HGS, HHO, WOA, PSO, SSA, PFA

In comparison experiment, to ensure the fairness in comparison, all algorithms are set with the same number of search agents (population size = 50) and the same number of function evaluations (max function evaluations = 100000). The number of dimensions for each benchmark function is 30-dimension. The specific parameter for each algorithm is selected based on default settings in the Mealpy library. One can change the parameters to adapt to their problems.

The experimental results of each model are produced by calculating the mean and standard deviation *std* of 10 trials running with the algorithms and functions mentioned above. The mean's results of tested algorithms are shown in Tables 11 and 12. Meanwhile, the *std*'s results are represented in Tables 13 and 14. In general, the results show that the algorithms in Mealpy achieve competitive performance overall on

[13] https://github.com/thieu1995/MHA-TSP
[14] https://github.com/thieu1995/MHA-FS

Table 11
The mean results after 10 trials of all tested optimizers on CEC-2017 benchmark functions.

Model	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
DE	1.56E+10	4.59E+04	1.04E+03	1.29E+04	2.79E−02	5.70E+04	5.09E+00	2.91E+01	4.82E+03	5.80E+04
CRO	7.30E+05	5.03E+00	3.32E+01	1.59E+02	3.22E−04	1.19E+03	2.18E−02	9.39E+00	3.59E+03	3.41E+04
GA	3.55E+06	9.51E+03	7.41E+01	2.01E+02	4.68E−05	4.83E+02	2.08E−02	2.11E+00	1.60E+03	1.79E+04
CGO	3.64E+10	4.69E+04	7.52E+03	2.77E+04	6.94E−03	1.59E+04	4.28E−01	1.86E+01	5.63E+03	1.07E+05
GBO	4.36E+08	2.11E+03	1.36E+02	8.84E+02	5.02E−04	3.25E+04	2.91E−02	1.13E+01	5.21E+03	9.57E+04
PSS	3.24E+09	8.41E+03	3.60E+02	4.47E+03	1.21E−02	3.17E+04	1.56E+00	5.24E+00	6.71E+03	2.17E+05
SMA	3.58E+05	3.81E+00	1.06E+02	3.49E+02	4.17E−02	3.35E+04	1.07E+00	3.71E+00	3.31E+03	6.83E+03
EOA	3.50E+10	3.81E+04	3.74E+03	5.70E+04	1.01E−02	2.26E+04	8.94E−01	2.19E+01	4.96E+03	1.27E+07
BBO	1.14E+07	4.75E+03	1.54E+02	3.05E+02	2.73E−04	5.42E+03	5.36E−02	2.82E+00	2.92E+03	2.39E+04
AEO	6.36E+09	2.39E+04	2.09E+03	1.67E+04	1.24E−02	1.30E+04	5.92E−01	8.44E+00	5.69E+03	1.32E+06
GCO	6.39E+09	5.19E+04	4.49E+02	9.97E+03	3.31E−02	7.26E+04	3.33E+00	1.50E+01	4.18E+03	1.31E+07
TLO	1.21E+03	2.32E−03	1.42E+02	1.83E+02	1.94E−04	4.58E+03	7.07E−03	5.08E+00	7.54E+03	3.95E+03
FBIO	1.21E+08	4.23E+02	2.46E+02	1.02E+03	1.99E−04	4.15E+03	1.50E−01	5.79E+00	3.54E+03	1.31E+04
GSKA	4.96E+03	1.91E−14	3.18E+01	1.84E+02	2.73E−03	8.32E+02	2.69E−01	5.44E−01	6.75E+03	1.23E+03
QSA	1.04E+04	1.23E+00	3.48E+01	5.76E+01	5.79E−03	8.28E+03	1.75E−03	5.34E+00	4.89E+03	2.03E+05
HS	9.70E+04	1.97E+00	3.19E+01	1.96E+02	4.39E−06	4.61E+03	1.96E−01	7.14E+00	2.91E+03	1.69E+04
HGSO	6.15E+10	1.01E+05	1.10E+04	8.54E+04	4.08E−02	9.85E+04	4.84E+00	5.14E+01	8.34E+03	7.76E+06
NRO	2.29E+07	1.33E+03	6.47E+01	2.55E+02	3.66E−03	2.07E+04	7.53E−01	8.95E+00	5.29E+03	3.76E+04
EFO	3.54E+10	1.40E+05	2.83E+03	5.37E+04	1.07E−02	3.97E+04	1.11E+00	1.49E+01	6.04E+03	8.33E+05
EO	1.21E+03	7.77E−14	1.46E+02	1.02E+02	9.38E−07	1.23E+04	5.33E−03	2.90E+00	7.14E+03	2.17E+04
DO	7.95E+09	6.24E+04	2.04E+03	3.69E+04	3.50E−02	5.71E+04	2.86E+00	1.99E+01	8.13E+03	2.99E+06
GWO	1.92E+10	7.44E+04	6.26E+03	3.45E+04	1.12E−02	2.63E+04	1.72E+00	1.15E+01	5.45E+03	4.41E+05
HGS	3.37E+08	7.47E+03	6.46E+02	3.20E+04	6.07E−04	4.80E+03	1.66E−01	8.35E+00	6.69E+03	1.09E+05
HHO	5.05E+09	3.20E+04	4.65E+02	1.15E+04	6.56E−03	1.34E+04	1.87E−01	1.99E+01	3.01E+03	1.36E+05
WOA	5.00E+09	1.89E+04	2.60E+02	1.25E+04	3.67E−03	4.61E+03	4.65E−02	2.17E+01	7.29E+03	1.56E+05
PSO	8.25E+10	2.33E+05	1.42E+04	5.09E+04	1.01E−01	1.02E+05	7.61E+00	8.08E+01	8.19E+03	1.41E+07
SSA	4.51E+07	3.03E+03	3.10E+02	7.82E+03	5.84E−03	5.54E+03	3.58E−01	1.73E+01	6.34E+03	2.29E+05
PFA	3.50E+10	8.34E+04	3.35E+03	2.28E+04	3.59E−02	3.23E+04	3.16E+00	4.50E+01	7.69E+03	1.57E+07

Table 12
The mean results after 10 trials of all tested optimizers on CEC-2017 benchmark functions (continue).

Model	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20
DE	3.8E+08	5.1E+07	8.0E+04	4.3E+06	2.4E+04	4.1E+03	1.0E+05	1.5E+07	2.4E+03	2.0E+03
CRO	6.8E+04	1.1E+04	8.4E+05	3.0E+04	2.2E+01	1.4E+03	1.9E+05	5.2E+04	4.5E+02	3.7E+02
GA	3.2E+05	3.0E+05	1.4E+07	1.2E+05	1.4E+02	1.9E+03	4.3E+06	1.0E+06	1.3E+02	4.2E+02
CGO	3.1E+09	3.0E+09	3.5E+05	1.0E+08	8.2E+07	3.9E+08	8.0E+05	3.8E+10	5.8E+03	1.7E+04
GBO	9.6E+06	6.5E+06	1.7E+05	1.5E+05	1.5E+04	2.9E+03	8.6E+04	1.7E+05	1.1E+03	3.4E+02
PSS	4.0E+08	1.7E+08	7.1E+05	2.9E+07	1.7E+06	3.9E+05	1.3E+05	2.0E+08	1.3E+03	2.9E+03
SMA	1.1E+05	1.3E+04	8.7E+04	1.0E+04	1.2E+02	9.8E+02	1.3E+05	6.3E+04	2.6E+02	2.2E+02
EOA	5.5E+09	2.3E+09	1.2E+06	5.8E+08	1.4E+07	4.2E+07	1.7E+06	5.5E+09	3.1E+03	1.2E+04
BBO	8.4E+05	1.4E+06	2.0E+06	8.4E+05	1.6E+06	1.1E+03	1.1E+06	3.1E+06	3.1E+02	2.0E+02
AEO	3.0E+08	1.5E+08	4.7E+05	3.4E+07	5.0E+06	6.4E+04	1.2E+05	2.0E+10	5.0E+03	1.1E+04
GCO	1.1E+09	1.4E+07	1.3E+06	5.9E+07	1.8E+08	3.6E+04	1.4E+05	7.4E+09	1.0E+03	2.3E+03
TLO	1.1E+04	8.2E+03	1.4E+05	1.6E+04	2.3E+02	4.0E+02	8.4E+04	9.2E+03	3.0E+02	1.0E+02
FBIO	8.3E+06	5.8E+04	6.8E+04	4.1E+04	5.3E+02	1.4E+03	4.4E+04	1.2E+04	5.8E+02	2.2E+03
GSKA	5.5E+04	1.9E+03	1.1E+05	4.3E+04	1.1E+01	1.8E+03	1.1E+05	5.2E+04	2.6E+02	3.9E+02
QSA	1.1E+05	2.4E+03	4.9E+04	4.7E+04	7.7E+03	9.1E+02	1.4E+04	1.5E+04	5.2E+02	3.4E+02
HS	5.5E+04	5.5E+03	1.6E+06	1.1E+04	1.6E+01	2.4E+02	2.7E+04	2.5E+03	5.6E+01	4.8E+02
HGSO	1.0E+10	1.1E+10	8.2E+06	5.0E+09	1.8E+09	2.1E+12	8.1E+06	6.3E+12	5.4E+03	4.2E+04
NRO	3.8E+06	2.4E+05	4.6E+04	7.0E+04	1.9E+03	1.6E+04	3.7E+04	2.8E+04	1.2E+03	2.3E+02
EFO	2.7E+09	5.9E+09	6.4E+07	3.0E+09	5.8E+07	1.3E+09	1.1E+06	1.3E+11	6.1E+03	2.6E+04
EO	5.0E+03	9.3E+03	1.3E+05	1.2E+04	5.1E+02	1.3E+03	1.9E+05	8.9E+03	3.2E+02	2.7E+02
DO	1.1E+09	1.1E+09	8.4E+06	3.4E+08	2.4E+07	2.6E+08	6.9E+06	1.5E+10	3.6E+03	5.6E+03
GWO	1.8E+09	9.0E+08	4.3E+06	7.1E+08	2.1E+08	2.8E+07	9.8E+04	5.6E+11	3.4E+03	3.2E+04
HGS	7.7E+06	3.3E+06	4.6E+05	3.9E+05	5.2E+03	5.8E+04	1.4E+05	7.8E+04	2.8E+03	1.1E+03
HHO	1.7E+07	3.3E+07	2.9E+05	3.0E+06	8.5E+04	2.6E+04	5.9E+04	1.3E+08	9.2E+03	6.3E+03
WOA	2.5E+06	3.1E+05	3.7E+05	2.2E+05	3.5E+05	3.5E+04	3.7E+04	1.1E+06	8.8E+03	1.1E+04
PSO	1.2E+10	2.0E+10	7.6E+06	9.3E+09	6.2E+08	6.7E+07	5.2E+06	1.9E+13	6.4E+03	1.0E+05
SSA	3.4E+06	5.5E+06	2.6E+05	1.1E+05	1.9E+04	5.7E+04	1.1E+05	5.1E+04	1.1E+04	5.3E+02
PFA	3.1E+09	2.2E+09	1.7E+07	5.1E+08	1.7E+08	8.2E+09	2.3E+06	1.8E+11	7.3E+03	2.8E+04

all benchmark functions. Also, as is observed in *std* results, the tested algorithms have stability characteristics.

We also present the convergence charts for various benchmark functions using the tested algorithms, as shown in Figs. 9, 10, 11, 12, 13, 14, and 15. It is important to note that some algorithms may have a lower number of iterations than others. As explained in Section 2.4, this is because certain algorithms update the fitness function k times the population size ($k * pop\_size$) in each iteration. Thus, the number of iterations required for these algorithms will be lower than others if $k \neq 1$. For instance, for the CRO and EFO algorithms under consideration, with a population size of 50 and 100000 function evaluations, the maximum number of iterations will be $100\,000/50 = 2000$. Conversely, for algorithms such as GSKA or HGS, which update each population twice in every iteration, the maximum number of iterations will be $100\,000/(50 * 2) = 1000$. From the figures presented, it is evident that all the tested algorithms have the potential to converge. However, some algorithms may not have reached the global optimal point due to several reasons. Firstly, the number of function evaluations may

Table 13
The std results after 10 trials of all tested optimizers on CEC-2017 benchmark functions.

Model	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
DE	4.02E−06	0.00E+00	0.00E+00	1.92E−12	3.66E−18	0.00E+00	9.36E−16	0.00E+00	0.00E+00	7.67E−12
CRO	0.00E+00	0.00E+00	7.49E−15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
GA	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	3.66E−18	4.68E−16	0.00E+00	0.00E+00
CGO	0.00E+00	0.00E+00	9.59E−13	0.00E+00	0.00E+00	1.92E−12	0.00E+00	0.00E+00	0.00E+00	1.53E−11
GBO	0.00E+00	0.00E+00	0.00E+00	1.20E−13	0.00E+00	7.67E−12	0.00E+00	1.87E−15	0.00E+00	0.00E+00
PSS	7.48E+08	1.91E+03	7.46E+01	8.84E+02	5.77E−03	1.18E+04	7.15E−01	2.12E+00	2.27E+02	7.11E+04
SMA	0.00E+00	4.68E−16	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
EOA	0.00E+00	7.67E−12	0.00E+00	7.67E−12	0.00E+00	0.00E+00	0.00E+00	0.00E+00	9.59E−13	1.96E−09
BBO	0.00E+00	9.59E−13	0.00E+00	5.99E−14	0.00E+00	0.00E+00	7.31E−18	4.68E−16	0.00E+00	0.00E+00
AEO	0.00E+00	3.83E−12	0.00E+00	0.00E+00	1.83E−18	1.92E−12	0.00E+00	1.87E−15	9.59E−13	0.00E+00
GCO	1.01E−06	0.00E+00	0.00E+00	1.92E−12	0.00E+00	1.53E−11	0.00E+00	0.00E+00	9.59E−13	1.96E−09
TLO	0.00E+00	0.00E+00	0.00E+00	0.00E+00	2.86E−20	0.00E+00	0.00E+00	9.36E−16	9.59E−13	4.79E−13
FBIO	0.00E+00	5.99E−14	0.00E+00	1.20E−13	0.00E+00	0.00E+00	0.00E+00	9.36E−16	4.79E−13	1.92E−12
GSKA	9.59E−13	0.00E+00	7.49E−15	3.00E−14	4.57E−19	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
QSA	0.00E+00	2.34E−16	7.49E−15	7.49E−15	0.00E+00	1.92E−12	2.29E−19	0.00E+00	0.00E+00	0.00E+00
HS	0.00E+00	2.34E−16	0.00E+00	3.00E−14	0.00E+00	0.00E+00	0.00E+00	9.36E−16	0.00E+00	3.83E−12
HGSO	0.00E+00	0.00E+00	1.92E−12	0.00E+00	7.31E−18	1.53E−11	0.00E+00	0.00E+00	0.00E+00	9.82E−10
NRO	3.93E−09	2.40E−13	0.00E+00	3.00E−14	4.57E−19	3.83E−12	0.00E+00	0.00E+00	9.59E−13	0.00E+00
EFO	8.04E−06	0.00E+00	0.00E+00	7.67E−12	1.83E−18	0.00E+00	2.34E−16	1.87E−15	9.59E−13	0.00E+00
EO	0.00E+00	0.00E+00	3.00E−14	0.00E+00	2.23E−22	1.92E−12	0.00E+00	0.00E+00	0.00E+00	0.00E+00
DO	1.01E−06	0.00E+00	0.00E+00	7.67E−12	0.00E+00	0.00E+00	0.00E+00	0.00E+00	9.59E−13	0.00E+00
GWO	4.02E−06	1.53E−11	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	6.14E−11
HGS	0.00E+00	0.00E+00	1.20E−13	0.00E+00	0.00E+00	0.00E+00	2.93E−17	1.87E−15	9.59E−13	1.53E−11
HHO	0.00E+00	3.83E−12	0.00E+00	1.92E−12	9.14E−19	0.00E+00	0.00E+00	0.00E+00	4.79E−13	0.00E+00
WOA	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	9.59E−13	7.31E−18	0.00E+00	1.92E−12	3.07E−11
PSO	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.46E−17	0.00E+00	9.36E−16	1.50E−14	1.92E−12	1.96E−09
SSA	7.85E−09	0.00E+00	0.00E+00	9.59E−13	9.14E−19	0.00E+00	5.85E−17	0.00E+00	0.00E+00	3.07E−11
PFA	0.00E+00	1.53E−11	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.68E−16	0.00E+00	1.92E−12	1.96E−09

Table 14
The std results after 10 trials of all tested optimizers on CEC-2017 benchmark functions (continue).

Model	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20
DE	6.28E−08	0.00E+00	0.00E+00	9.82E−10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
CRO	0.00E+00	0.00E+00	1.23E−10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	7.67E−12	0.00E+00	0.00E+00
GA	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.23E−10	0.00E+00	5.99E−14
CGO	5.03E−07	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	9.59E−13	3.83E−12
GBO	1.96E−09	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.79E−13	0.00E+00	3.07E−11	2.40E−13	5.99E−14
PSS	6.80E+07	6.11E+07	4.50E+05	1.09E+07	1.87E+06	5.77E+05	5.08E+04	9.38E+07	4.66E+02	5.36E+02
SMA	0.00E+00	0.00E+00	1.53E−11	0.00E+00	1.50E−14	1.20E−13	0.00E+00	7.67E−12	0.00E+00	0.00E+00
EOA	1.01E−06	5.03E−07	2.45E−10	1.26E−07	1.96E−09	0.00E+00	2.45E−10	1.01E−06	0.00E+00	0.00E+00
BBO	1.23E−10	0.00E+00	2.45E−10	1.23E−10	2.45E−10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	3.00E−14
AEO	0.00E+00	0.00E+00	0.00E+00	0.00E+00	9.82E−10	0.00E+00	0.00E+00	4.02E−06	9.59E−13	0.00E+00
GCO	0.00E+00	0.00E+00	0.00E+00	7.85E−09	3.14E−08	7.67E−12	3.07E−11	1.01E−06	0.00E+00	4.79E−13
TLO	0.00E+00	1.92E−12	3.07E−11	0.00E+00	3.00E−14	5.99E−14	0.00E+00	0.00E+00	0.00E+00	0.00E+00
FBIO	9.82E−10	7.67E−12	0.00E+00	0.00E+00	0.00E+00	2.40E−13	7.67E−12	1.92E−12	0.00E+00	4.79E−13
GSKA	0.00E+00	2.40E−13	0.00E+00	0.00E+00	1.87E−15	0.00E+00	1.53E−11	7.67E−12	0.00E+00	0.00E+00
QSA	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.20E−13	1.92E−12	1.92E−12	0.00E+00	5.99E−14
HS	0.00E+00	9.59E−13	2.45E−10	0.00E+00	3.74E−15	0.00E+00	3.83E−12	4.79E−13	7.49E−15	5.99E−14
HGSO	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	9.82E−10	0.00E+00	9.59E−13	7.67E−12
NRO	4.91E−10	3.07E−11	0.00E+00	1.53E−11	2.40E−13	0.00E+00	7.67E−12	0.00E+00	0.00E+00	3.00E−14
EFO	5.03E−07	1.01E−06	7.85E−09	0.00E+00	0.00E+00	2.51E−07	0.00E+00	0.00E+00	9.59E−13	0.00E+00
EO	9.59E−13	1.92E−12	3.07E−11	0.00E+00	5.99E−14	0.00E+00	0.00E+00	0.00E+00	0.00E+00	5.99E−14
DO	2.51E−07	0.00E+00	0.00E+00	0.00E+00	0.00E+00	6.28E−08	0.00E+00	0.00E+00	0.00E+00	9.59E−13
GWO	2.51E−07	0.00E+00	9.82E−10	0.00E+00	3.14E−08	3.93E−09	0.00E+00	0.00E+00	0.00E+00	0.00E+00
HGS	0.00E+00	4.91E−10	6.14E−11	0.00E+00	0.00E+00	0.00E+00	3.07E−11	0.00E+00	0.00E+00	0.00E+00
HHO	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
WOA	4.91E−10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.92E−12
PSO	2.01E−06	0.00E+00	0.00E+00	2.01E−06	1.26E−07	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
SSA	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.53E−11	0.00E+00	0.00E+00	0.00E+00
PFA	0.00E+00	5.03E−07	3.93E−09	0.00E+00	3.14E−08	1.01E−06	0.00E+00	3.22E−05	9.59E−13	0.00E+00

not have been sufficient, and the population size may have been too small for effective searching. Secondly, the benchmark functions CEC-2017 are complex problems with multiple local optimal and concave points, making it challenging for the algorithms to identify the global optimal point. Thirdly, the design of the original algorithm may not have been adequate or complex enough to enable the algorithm to identify the global optimum. Finally, the algorithm's parameters may not have been appropriately tuned for each function. However, as the primary aim of this experiment was to demonstrate the implementation of the algorithms in Mealpy, we did not dedicate a significant amount of time to tune each algorithm's parameters for each problem.

3.2. Numerical results of parallelism experiment

In this section, we aim to demonstrate the effectiveness of various learning modes of the algorithm in Mealpy. We conduct two experiments using the WOA algorithm to solve two common problems. The first experiment involves solving the sphere benchmark function, while the second experiment focuses on tuning the hyper-parameters of the SVC model for the breast cancer classification problem [157]. The experiments were carried out on Ubuntu Server 20.04, with 12 GB RAM and 8 CPU cores.

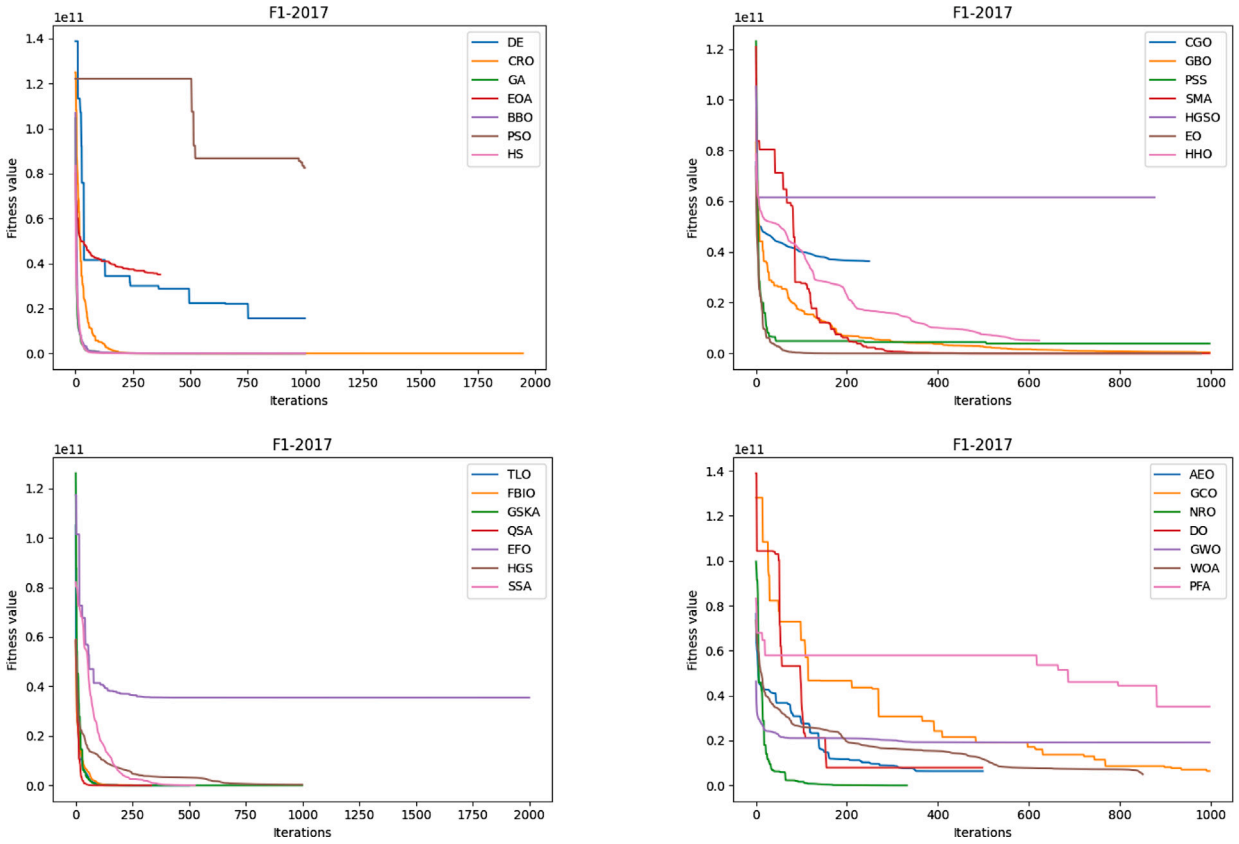


Fig. 9. The convergence chart of tested optimizers on F1-2017 function.

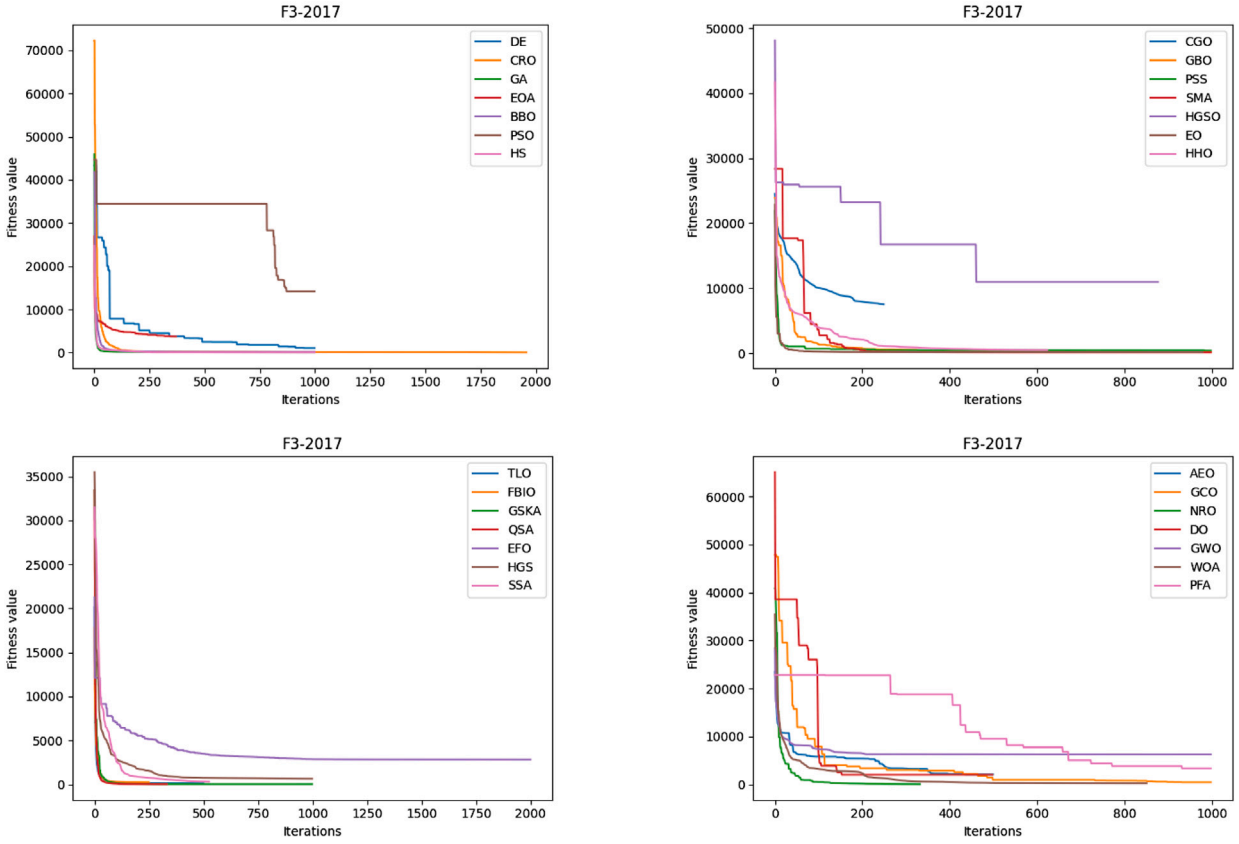


Fig. 10. The convergence chart of tested optimizers on F3-2017 function.

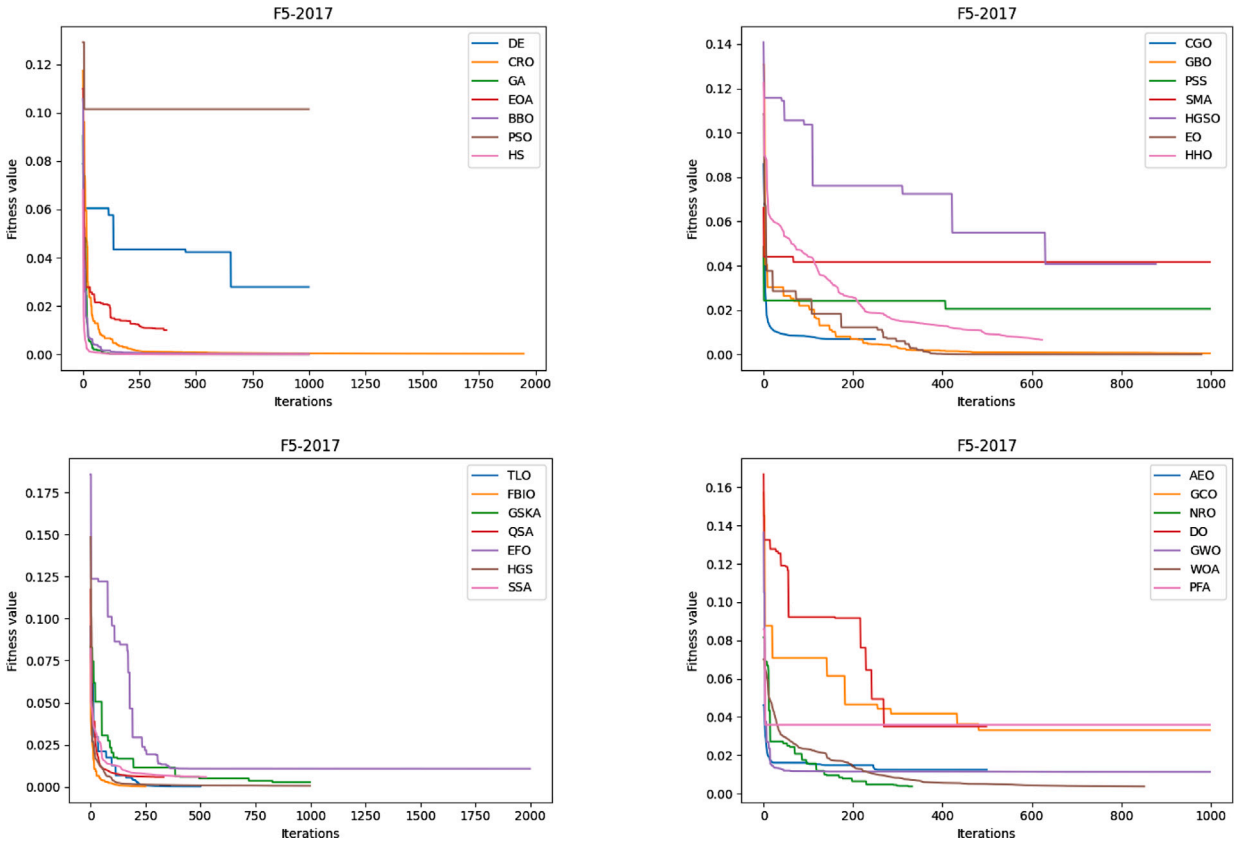


Fig. 11. The convergence chart of tested optimizers on F5-2017 function.

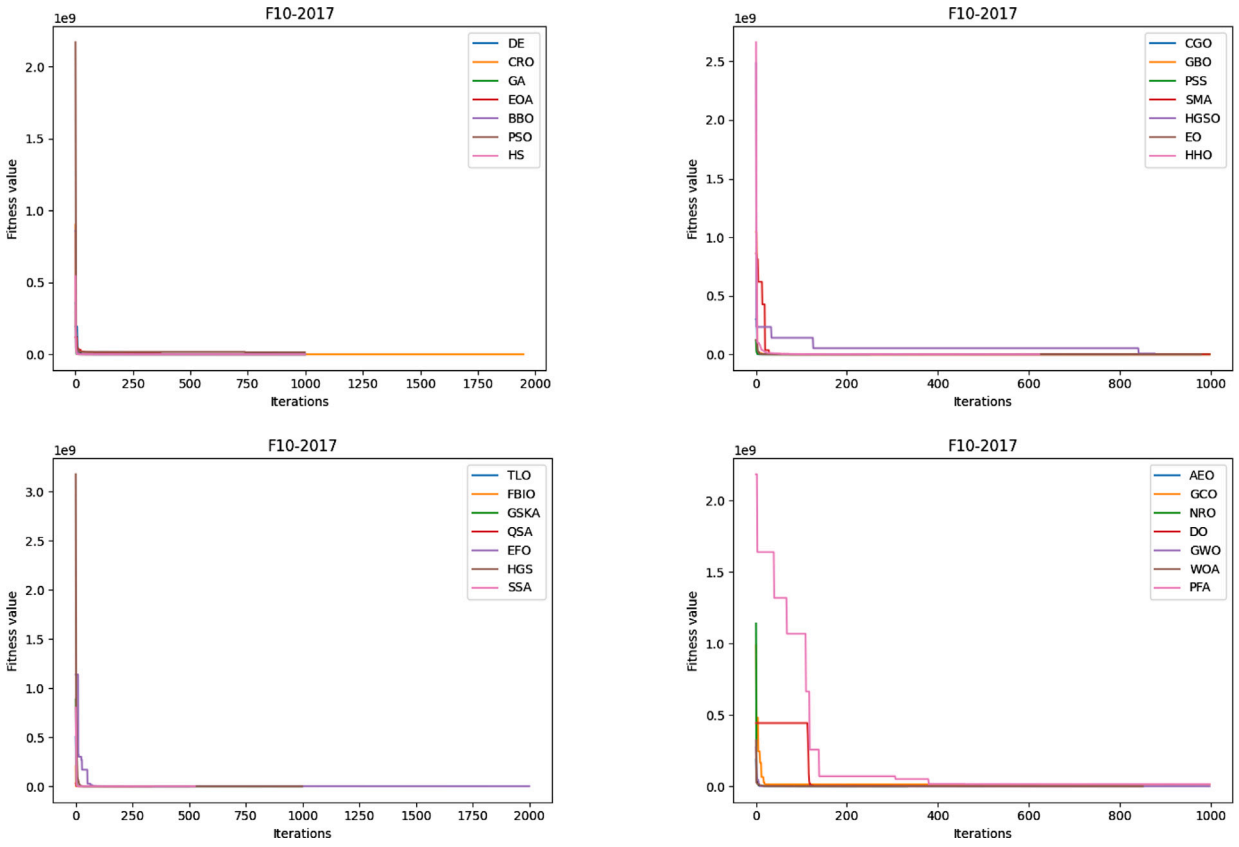


Fig. 12. The convergence chart of tested optimizers on F10-2017 function.

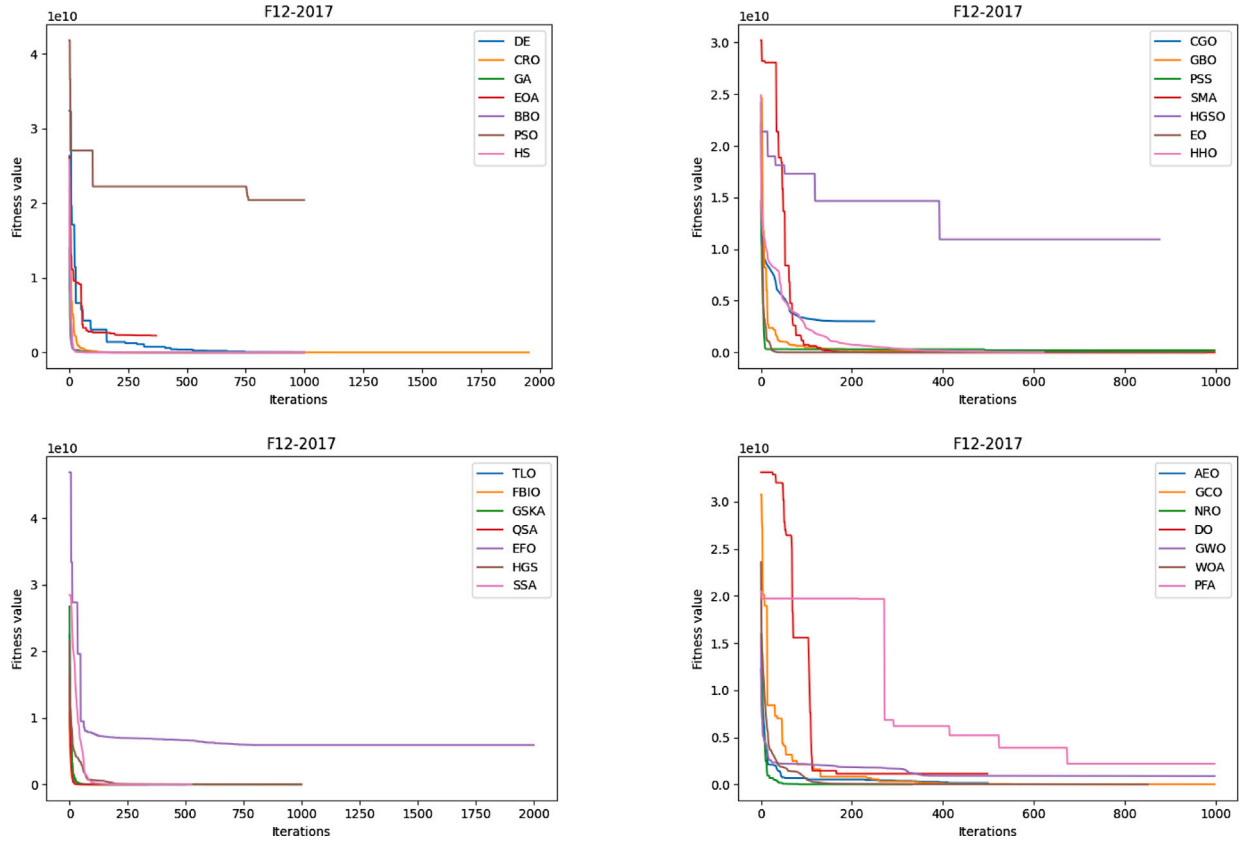


Fig. 13. The convergence chart of tested optimizers on F12-2017 function.

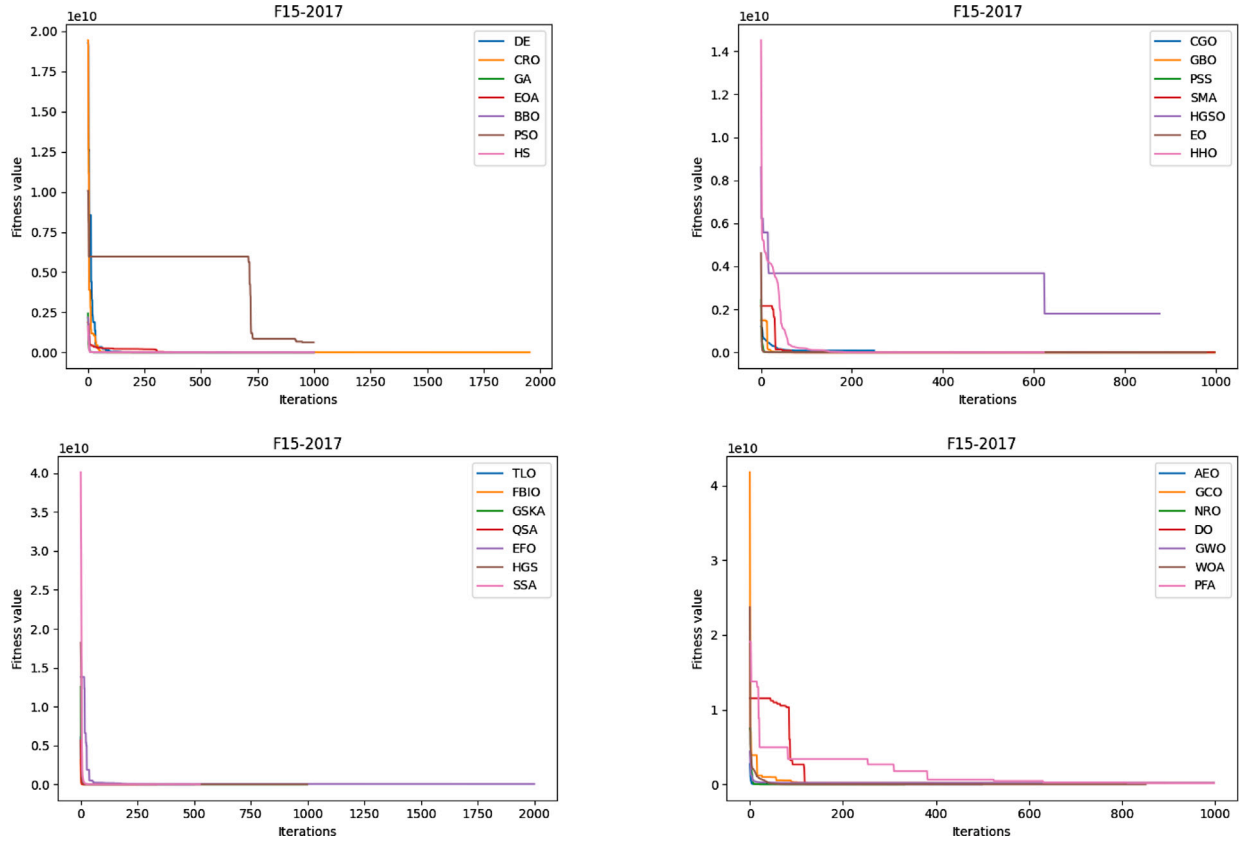


Fig. 14. The convergence chart of tested optimizers on F15-2017 function.

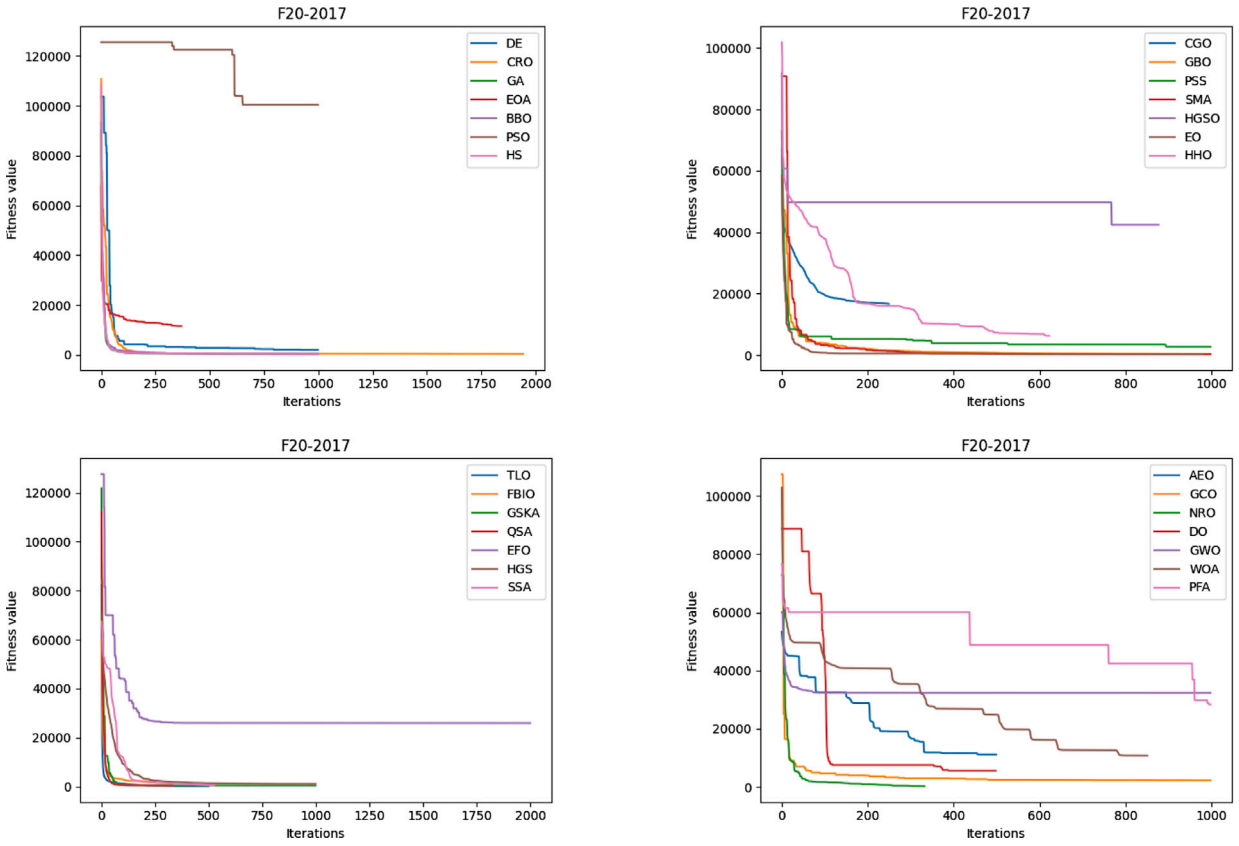


Fig. 15. The convergence chart of tested optimizers on F20-2017 function.

Table 15
The total run time (seconds) and best fitness value of WOA algorithm with different training mode on sphere benchmark function.

Trial	Total run time (seconds)				Best fitness value			
	Single	Swarm	Thread	Process	Single	Swarm	Thread	Process
1	1.3896	0.9447	1.7326	161.2338	3.7E−103	3.2E−102	2.2E−102	2.0E−101
2	1.3689	0.9440	1.6980	164.0105	2.1E−104	3.8E−102	1.1E−100	1.9E−106
3	1.3635	0.9512	1.6936	165.1846	9.1E−106	6.8E−106	3.6E−104	5.2E−105
4	1.3767	0.9592	1.7183	164.3560	1.1E−98	2.0E−103	6.4E−99	3.3E−99
5	1.3927	0.9575	1.6968	166.1221	3.2E−94	2.8E−103	1.2E−95	3.3E−102
6	1.3689	0.9646	1.6974	162.7452	1.1E−98	2.7E−100	1.9E−94	1.1E−97
7	1.3650	0.9632	1.7184	160.7158	1.1E−99	9.4E−98	6.0E−100	6.9E−101
8	1.3855	0.9515	1.6956	162.7990	1.2E−95	4.9E−103	1.5E−94	9.1E−98
9	1.3641	0.9542	1.6963	163.7677	1.9E−104	1.4E−100	3.7E−106	3.5E−99
10	1.3630	0.9499	1.7220	165.8856	2.4E−95	1.1E−101	6.2E−102	4.2E−101

Table 16
The total run time (seconds) and best fitness value (accuracy) of WOA algorithm with different training mode on tuning hyper-parameter of SVC model.

Trial	Total run time (seconds)				Best fitness value (accuracy)			
	Single	Swarm	Thread	Process	Single	Swarm	Thread	Process
1	6.4760	6.0586	4.0688	6.9525	0.97203	0.97203	0.97203	0.97203
2	5.8187	21.3267	3.5790	6.7010	0.97203	0.97203	0.97203	0.97203
3	6.2459	6.2242	3.6441	7.0737	0.97203	0.97203	0.97203	0.97203
4	5.9782	6.4469	3.8679	7.1050	0.97203	0.97203	0.97203	0.97203
5	5.8698	6.9215	4.1103	7.2111	0.97203	0.97203	0.97203	0.97203
6	5.6649	6.4986	3.7231	7.2022	0.97203	0.97203	0.97203	0.97203
7	5.7609	28.2068	3.7542	7.3750	0.97203	0.97203	0.97203	0.97203
8	6.7974	6.0424	3.8011	7.1949	0.97203	0.97203	0.97203	0.97203
9	5.8806	5.9132	3.7985	6.9623	0.97203	0.97203	0.97203	0.97203
10	5.7153	6.2446	3.5069	7.0110	0.97203	0.97203	0.97203	0.97203

We chose the sphere function problem for its simplicity and quick computation time, which takes only a few milliseconds to calculate the fitness function. However, when using parallel mode (multiple threads or multiple processes), the time to create a new thread or process to run in parallel is longer than the time to compute the fitness function. Therefore, the total execution time of the parallel mode is longer than that of the sequential mode, as shown in Table 15. The single and swarm modes have an average total run time of only 1.37 s and 0.95 s,

respectively, while thread parallelism has an average running time of around 1.7 s. The process mode has an average running time of up to 164 s, which is 120 times longer than the single mode and 172 times longer than the swarm mode. This is because creating a new process takes longer than creating a new thread. Additionally, due to the limited number of CPU cores in the test machine, new cores can only be created and continue to calculate fitness for other solutions after the old cores complete their work and are destroyed.

On the other hand, in the hyper-parameter tuning experiment for the SVC model, we need to train the SVC model with a new set of parameters on the training set and return the accuracy of the validation set as a fitness value. Therefore, the execution time can be up to seconds, which is much longer than calculating the sphere function. As shown in Table 15, the average processing time for sequential modes (single and swarm) is around 6 s. However, we observe a significant reduction in the processing time of approximately 3.8 s with thread parallelism mode. The process mode takes only around 7.2 s.

In conclusion, when the problem is more complex, such as optimizing the hyper-parameters of a neural network model, the time to calculate the fitness function will be much longer. In this case, using parallelization mode (thread or process) can provide a much faster solution than sequential modes (single or swarm). It is important to note that the best fitness value results were almost unchanged for different learning modes, as evidenced in Table 14 and specifically in Table 15, where all learning modes have the same best fitness values (accuracy).

Therefore, we can conclude that if the problem is more significant, for example, optimizing the neural network model's hyper-parameter, it will take much more time to train and test, so the time to calculate the fitness function will be much longer. Using parallelization mode (thread or process) will help the solver to give a much faster solution than sequential modes (single or swarm).

It is worth noting that the best fitness value results were almost unchanged for different learning regimens. It can be seen clearly in Table 15 and, more specifically, in Table 16. All learning modes' best fitness values (accuracy) are the same.

4. Conclusions and future work

Considerable effort and attention have been invested in creating a library that unites all the Meta-Heuristic Algorithms (MHAs) in a modular and user-friendly manner. Despite the availability of several MHA libraries, they only guarantee some of the essential requirements for users, such as comprehensive documentation, extensive examples, easy-to-use visualization modules, parallelism modes, and straightforward implementations for better transparency.

This paper presents an open-source and cross-platform library for MHAs called Mealpy. Utilizing the features of Object-Oriented Programming (OOP) in combination with Python, Mealpy provides and fulfills all the above requirements. Furthermore, Mealpy can be integrated with popular libraries like Tensorflow, Keras, PyTorch, and Scikit-learn. Essentially, for any problem that can be formulated as an optimization problem, Mealpy can act as a black box and solve it. Our goal with Mealpy is to implement all the original MHAs, with over 160 algorithms currently available from different inspiration sources such as evolutionary-based, swarm-based, human-based, physics-based, bio-based, math-based, system-based, and music-based. Moreover, the significant difference between Mealpy and other libraries is that it has up to four training modes, including two parallelization modes, which assist users in increasing processing speed for large-scale problems without altering or re-implementing the algorithm in a parallelizable form.

Additionally, Mealpy provides solutions for multi-objective, constrained, and even discrete problems, significantly increasing its usability in solving real-world problems. Finally, Mealpy supports history-based objects that track the optimization process and graphics content, such as convergence charts, runtime charts, population diversity charts, or exploration versus exploitation charts. Based on these visualization figures, users can evaluate and conclude a tested algorithm for a specific problem.

In future work, we plan to continue implementing more meta-heuristic algorithms and support different multi-objective methods and techniques for constrained problems.

CRediT authorship contribution statement

Nguyen Van Thieu: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Seyedali Mirjalili:** Resources, Validation, Writing – original draft, Writing – review & editing, Supervision.

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.

Data availability

Data will be made available on request.

References

[1] S.E. De Leon-Aldaco, H. Calleja, J. Aguayo Alquicira, Metaheuristic optimization methods applied to power converters: A review, IEEE Trans. Power Electron. 30 (12) (2015) 6791–6803, http://dx.doi.org/10.1109/TPEL.2015.2397311, URL: http://ieeexplore.ieee.org/document/7024140/.

[2] R.E. Neapolitan, K. Naimipour, Foundations of Algorithms using Java Pseudocode, Jones and Bartlett Publishers, Sudbury, Mass, 2004, OCLC: ocm53138748.

[3] T.E. Mallouk, Divide and conquer, Nature Chem. 5 (5) (2013) 362–363, http://dx.doi.org/10.1038/nchem.1634, URL: http://www.nature.com/articles/nchem.1634.

[4] D.P. de Farias, B. Van Roy, The linear programming approach to approximate dynamic programming, Oper. Res. 51 (6) (2003) 850–865, http://dx.doi.org/10.1287/opre.51.6.850.24925, URL: http://pubsonline.informs.org/doi/abs/10.1287/opre.51.6.850.24925.

[5] D.R. Morrison, S.H. Jacobson, J.J. Sauppe, E.C. Sewell, Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning, Discrete Optim. 19 (2016) 79–102, http://dx.doi.org/10.1016/j.disopt.2016.01.005, URL: https://linkinghub.elsevier.com/retrieve/pii/S1572528616000062.

[6] P. Civicioglu, Backtracking search optimization algorithm for numerical optimization problems, Appl. Math. Comput. 219 (15) (2013) 8121–8144, http://dx.doi.org/10.1016/j.amc.2013.02.017, URL: https://linkinghub.elsevier.com/retrieve/pii/S0096300313001380.

[7] Z. Beheshti, S.M.H. Shamsuddin, A review of population-based meta-heuristic algorithms, Int. J. Adv. Soft Comput. Appl. 5 (1) (2013) 1–35.

[8] S. Talatahari, H. Bayzidi, M. Saraee, Social network search for global optimization, IEEE Access 9 (2021) 92815–92863, http://dx.doi.org/10.1109/ACCESS.2021.3091495, URL: https://ieeexplore.ieee.org/document/9462076/.

[9] A.N. Ahmed, T. Van Lam, N.D. Hung, N. Van Thieu, O. Kisi, A. El-Shafie, A comprehensive comparison of recent developed meta-heuristic algorithms for streamflow time series forecasting problem, Appl. Soft Comput. 105 (2021) 107282, http://dx.doi.org/10.1016/j.asoc.2021.107282, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494621002052.

[10] P.J.M. van Laarhoven, E.H.L. Aarts, Simulated Annealing: Theory and Applications, Springer Netherlands, Dordrecht, 1987, http://dx.doi.org/10.1007/978-94-015-7744-1, URL: http://link.springer.com/10.1007/978-94-015-7744-1.

[11] D. Whitley, A genetic algorithm tutorial, Stat. Comput. 4 (2) (1994) 65–85.

[12] R. Mallipeddi, P.N. Suganthan, Q.-K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2) (2011) 1679–1696, http://dx.doi.org/10.1016/j.asoc.2010.04.024.

[13] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4, IEEE, Perth, WA, Australia, 1995, pp. 1942–1948, http://dx.doi.org/10.1109/ICNN.1995.488968, URL: http://ieeexplore.ieee.org/document/488968/.

[14] K. Socha, M. Dorigo, Ant colony optimization for continuous domains, European J. Oper. Res. 185 (3) (2008) 1155–1173, http://dx.doi.org/10.1016/j.ejor.2006.06.046, URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221706006333.

[15] S. Mirjalili, A. Lewis, The whale optimization algorithm, Adv. Eng. Softw. 95 (2016) 51–67, http://dx.doi.org/10.1016/j.advengsoft.2016.01.008, URL: https://linkinghub.elsevier.com/retrieve/pii/S0965997816300163.

[16] Y. Ho, D. Pepyne, Simple explanation of the no-free-lunch theorem and its implications, J. Optim. Theory Appl. 115 (3) (2002) 549–570, http://dx.doi.org/10.1023/A:1021251113462, URL: http://link.springer.com/10.1023/A:1021251113462.

[17] H. Faris, I. Aljarah, S. Mirjalili, P.A. Castillo, J.J.M. Guervós, EvoloPy: An open-source nature-inspired optimization framework in python., Int. J. Child-Comput. Interact. (ECTA) 1 (2016) 171–177, URL: https://github.com/7ossam81/EvoloPy.

[18] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: Evolutionary algorithms made easy, J. Mach. Learn. Res. 13 (2012) 2171–2175, URL: https://github.com/DEAP/deap.

[19] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, J. Portilla-Figueras, The coral reefs optimization algorithm: a novel metaheuristic for efficiently solving optimization problems, Sci. World J. 2014 (2014) http://dx.doi.org/10.1155/2014/739768.

[20] T. Nguyen, T. Nguyen, B.M. Nguyen, G. Nguyen, Efficient time-series forecasting using neural network and opposition-based coral reefs optimization, Int. J. Comput. Intell. Syst. 12 (2) (2019) 1144, http://dx.doi.org/10.2991/ijcis.d.190930.003.

[21] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (5) (2009) 945–958, http://dx.doi.org/10.1109/TEVC.2009.2014613.

[22] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 2, IEEE, 2005, pp. 1785–1791, http://dx.doi.org/10.1109/CEC.2005.1554904.

[23] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 71–78, http://dx.doi.org/10.1109/CEC.2013.6557555.

[24] R. Tanabe, A.S. Fukunaga, Improving the search performance of SHADE using linear population size reduction, in: 2014 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2014, pp. 1658–1665, http://dx.doi.org/10.1109/CEC.2014.6900380.

[25] J. Teo, Exploring dynamic self-adaptive populations in differential evolution, Soft Comput. 10 (8) (2006) 673–686, http://dx.doi.org/10.1007/s00500-005-0537-1.

[26] H.-G. Beyer, H.-P. Schwefel, Evolution strategies–a comprehensive introduction, Nat. Comput. 1 (1) (2002) 3–52, http://dx.doi.org/10.1023/A:1015059928466.

[27] E. Salari, Competitive learning vector quantization with evolution strategies for image compression, Opt. Eng. 44 (2) (2005) 027006, http://dx.doi.org/10.1117/1.1839892, URL: http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/1.1839892.

[28] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Trans. Evol. Comput. 3 (2) (1999) 82–102, http://dx.doi.org/10.1109/4235.771163.

[29] C.-Y. Lee, X. Yao, Evolutionary algorithms with adaptive Levy mutations, in: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), Vol. 1, IEEE, Seoul, South Korea, 2001, pp. 568–575, http://dx.doi.org/10.1109/CEC.2001.934442, URL: http://ieeexplore.ieee.org/document/934442/.

[30] X.-S. Yang, Flower pollination algorithm for global optimization, in: International Conference on Unconventional Computing and Natural Computation, Springer, 2012, pp. 240–249, http://dx.doi.org/10.1007/978-3-642-32894-7_27.

[31] P. Moscato, et al., On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Caltech Concurrent Computation Program, C3P Report 826, Citeseer, 1989, p. 1989.

[32] B. Abdollahzadeh, F.S. Gharehchopogh, S. Mirjalili, African vultures optimization algorithm: A new nature-inspired metaheuristic algorithm for global optimization problems, Comput. Ind. Eng. 158 (2021) 107408, http://dx.doi.org/10.1016/j.cie.2021.107408.

[33] S. Mirjalili, The ant lion optimizer, Adv. Eng. Softw. 83 (2015) 80–98, http://dx.doi.org/10.1016/j.advengsoft.2015.01.010, URL: https://linkinghub.elsevier.com/retrieve/pii/S0965997815000113.

[34] L. Abualigah, D. Yousri, M. Abd Elaziz, A.A. Ewees, M.A. Al-qaness, A.H. Gandomi, Aquila Optimizer: A novel meta-heuristic optimization algorithm, Comput. Ind. Eng. 157 (2021) 107250, http://dx.doi.org/10.1016/j.cie.2021.107250, URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835221001546.

[35] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput. 8 (1) (2008) 687–697, http://dx.doi.org/10.1016/j.asoc.2007.05.007, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568946070000531.

[36] B. Abdollahzadeh, F. Soleimanian Gharehchopogh, S. Mirjalili, Artificial gorilla troops optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems, Int. J. Intell. Syst. 36 (10) (2021) 5887–5958, http://dx.doi.org/10.1002/int.22535.

[37] L. Wang, Q. Cao, Z. Zhang, S. Mirjalili, W. Zhao, Artificial rabbits optimization: A new bio-inspired meta-heuristic algorithm for solving engineering optimization problems, Eng. Appl. Artif. Intell. 114 (2022) 105082, http://dx.doi.org/10.1016/j.engappai.2022.105082.

[38] K.M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Syst. Mag. 22 (3) (2002) 52–67, http://dx.doi.org/10.1109/MCS.2002.1004010, URL: https://ieeexplore.ieee.org/document/1004010/.

[39] T. Nguyen, B.M. Nguyen, G. Nguyen, Building resource auto-scaler with functional-link neural network and adaptive bacterial foraging optimization, in: T. Gopal, J. Watada (Eds.), Theory and Applications of Models of Computation, vol. 11436, Springer International Publishing, Cham, 2019, pp. 501–517, http://dx.doi.org/10.1007/978-3-030-14812-6_31, URL: http://link.springer.com/10.1007/978-3-030-14812-6_31.

[40] H.A. Alsattar, A.A. Zaidan, B.B. Zaidan, Novel meta-heuristic bald eagle search optimisation algorithm, Artif. Intell. Rev. 53 (3) (2020) 2237–2264, http://dx.doi.org/10.1007/s10462-019-09732-5, URL: http://link.springer.com/10.1007/s10462-019-09732-5.

[41] X.-S. Yang, A new metaheuristic bat-inspired algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), Vol. 284, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 65–74, http://dx.doi.org/10.1007/978-3-642-12538-6_6, URL: http://link.springer.com/10.1007/978-3-642-12538-6_6.

[42] X. Wang, W. Wang, Y. Wang, An adaptive bat algorithm, in: Intelligent Computing Theories and Technology, vol. 7996, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 216–223, http://dx.doi.org/10.1007/978-3-642-39482-9_25, URL: http://link.springer.com/10.1007/978-3-642-39482-9_25.

[43] N.S. Jaddi, S. Abdullah, A.R. Hamdan, Optimization of neural network model using modified bat-inspired algorithm, Appl. Soft Comput. 37 (2015) 71–86, http://dx.doi.org/10.1016/j.asoc.2015.08.002, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568946615004950.

[44] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, M. Zaidi, The Bees algorithm — A novel tool for complex optimisation problems, in: Intelligent Production Machines and Systems, Elsevier Science Ltd, Oxford, 2006, pp. 454–459, http://dx.doi.org/10.1016/B978-008045157-2/50081-X, URL: https://www.sciencedirect.com/science/article/pii/B978008045157250081X.

[45] D.T. Pham, M. Castellani, A comparative study of the Bees Algorithm as a tool for function optimisation, in: J. Chen (Ed.), Cogent Eng. 2 (1) (2015) 1091540, http://dx.doi.org/10.1080/23311916.2015.1091540, URL: https://www.tandfonline.com/doi/full/10.1080/23311916.2015.1091540.

[46] X.-B. Meng, X. Gao, L. Lu, Y. Liu, H. Zhang, A new bio-inspired optimisation algorithm: Bird Swarm Algorithm, J. Exp. Theor. Artif. Intell. 28 (4) (2016) 673–687, http://dx.doi.org/10.1080/0952813X.2015.1042530, URL: http://www.tandfonline.com/doi/full/10.1080/0952813X.2015.1042530.

[47] S.-C. Chu, P.-w. Tsai, J.-S. Pan, Cat swarm optimization, in: Q. Yang, G. Webb (Eds.), PRICAI 2006: Trends in Artificial Intelligence, Springer, Berlin, Heidelberg, 2006, pp. 854–858, http://dx.doi.org/10.1007/978-3-540-36668-3_94.

[48] J. Pierezan, L. Dos Santos Coelho, Coyote optimization algorithm: A new metaheuristic for global optimization problems, in: 2018 IEEE Congress on Evolutionary Computation, CEC, 2018, pp. 1–8, http://dx.doi.org/10.1109/CEC.2018.8477769.

[49] X.-S. Yang, S. Deb, Cuckoo search via Lévy flights, in: 2009 World Congress on Nature & Biologically Inspired Computing, NaBIC, 2009, pp. 210–214, http://dx.doi.org/10.1109/NABIC.2009.5393690.

[50] S. Mirjalili, Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems, Neural Comput. Appl. 27 (4) (2016) 1053–1073, http://dx.doi.org/10.1007/s00521-015-1920-1, URL: http://link.springer.com/10.1007/s00521-015-1920-1.

[51] J.O. Agushaka, A.E. Ezugwu, L. Abualigah, Dwarf mongoose optimization algorithm, Comput. Methods Appl. Mech. Engrg. 391 (2022) 114570, http://dx.doi.org/10.1016/j.cma.2022.114570.

[52] G.-G. Wang, S. Deb, L.d.S. Coelho, Elephant herding optimization, in: 2015 3rd International Symposium on Computational and Business Intelligence, ISCBI, IEEE, Bali, Indonesia, 2015, pp. 1–5, http://dx.doi.org/10.1109/ISCBI.2015.8, URL: http://ieeexplore.ieee.org/document/7383528/.

[53] A.H. Gandomi, X.-S. Yang, A.H. Alavi, Mixed variable structural optimization using Firefly Algorithm, Comput. Struct. 89 (23–24) (2011) 2325–2336, http://dx.doi.org/10.1016/j.compstruc.2011.08.002, URL: https://linkinghub.elsevier.com/retrieve/pii/S0045794911002185.

[54] Y. Tan, Y. Zhu, Fireworks Algorithm for Optimization, in: Advances in Swarm Intelligence, Vol. 6145, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 355–364, http://dx.doi.org/10.1007/978-3-642-13495-1_44, URL: http://link.springer.com/10.1007/978-3-642-13495-1_44.

[55] W.-T. Pan, A new Fruit Fly Optimization Algorithm: Taking the financial distress model as an example, Knowl.-Based Syst. 26 (2012) 69–74, http://dx.doi.org/10.1016/j.knosys.2011.07.001, URL: https://linkinghub.elsevier.com/retrieve/pii/S0950705111001365.

[56] Y. Fan, P. Wang, A.A. Heidari, M. Wang, X. Zhao, H. Chen, C. Li, Boosted hunting-based fruit fly optimization and advances in real-world problems, Expert Syst. Appl. 159 (2020) 113502, http://dx.doi.org/10.1016/j.eswa.2020.113502, URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417420303262.

[57] S. Saremi, S. Mirjalili, A. Lewis, Grasshopper Optimisation Algorithm: Theory and application, Adv. Eng. Softw. 105 (2017) 30–47, http://dx.doi.org/10.1016/j.advengsoft.2017.01.004, URL: https://www.sciencedirect.com/science/article/pii/S0965997816305646.

[58] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey Wolf Optimizer, Adv. Eng. Softw. 69 (2014) 46–61, http://dx.doi.org/10.1016/j.advengsoft.2013.12.007, URL: https://linkinghub.elsevier.com/retrieve/pii/S0965997813001853.

[59] S. Gupta, K. Deep, A novel Random Walk Grey Wolf Optimizer, Swarm Evol. Comput. 44 (2019) 101–112, http://dx.doi.org/10.1016/j.swevo.2018.01.001, URL: https://linkinghub.elsevier.com/retrieve/pii/S2210650217305333.

[60] O.O. Obadina, M.A. Thaha, K. Althoefer, M.H. Shaheed, Dynamic characterization of a master–slave robotic manipulator using a hybrid grey wolf–whale optimization algorithm, J. Vib. Control (2021) 10775463211003402, http://dx.doi.org/10.1177/10775463211003402.

[61] A.A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, H. Chen, Harris hawks optimization: Algorithm and applications, Future Gener. Comput. Syst. 97 (2019) 849–872, http://dx.doi.org/10.1016/j.future.2019.02.028, URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X18313530.

[62] F.A. Hashim, E.H. Houssein, K. Hussain, M.S. Mabrouk, W. Al-Atabany, Honey Badger Algorithm: New metaheuristic algorithm for solving optimization problems, Math. Comput. Simulation 192 (2022) 84–110, http://dx.doi.org/10.1016/j.matcom.2021.08.013.

[63] Y. Yang, H. Chen, A.A. Heidari, A.H. Gandomi, Hunger games search: Visions, conception, implementation, deep analysis, perspectives, and towards performance shifts, Expert Syst. Appl. 177 (2021) 114864, http://dx.doi.org/10.1016/j.eswa.2021.114864, URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417421003055.

[64] R. Venkata Rao, Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems, Int. J. Ind. Eng. Comput. (2016) 19–34, http://dx.doi.org/10.5267/j.ijiec.2015.8.004, URL: http://www.growingscience.com/ijiec/Vol7/IJIEC_2015_32.pdf.

[65] G. Iacca, V.C. dos Santos Junior, V. Veloso de Melo, An improved Jaya optimization algorithm with Lévy flight, Expert Syst. Appl. 165 (2021) 113902, http://dx.doi.org/10.1016/j.eswa.2020.113902, URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417420306989.

[66] W. Zhao, Z. Zhang, L. Wang, Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications, Eng. Appl. Artif. Intell. 87 (2020) 103300, http://dx.doi.org/10.1016/j.engappai.2019.103300, URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197619302593.

[67] A. Faramarzi, M. Heidarinejad, S. Mirjalili, A.H. Gandomi, Marine predators algorithm: A nature-inspired metaheuristic, Expert Syst. Appl. 152 (2020) 113377, http://dx.doi.org/10.1016/j.eswa.2020.113377.

[68] S. Mirjalili, Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm, Knowl.-Based Syst. 89 (2015) 228–249, http://dx.doi.org/10.1016/j.knosys.2015.07.006, URL: https://linkinghub.elsevier.com/retrieve/pii/S0950705115002580.

[69] G.-G. Wang, Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems, Memet. Comput. 10 (2) (2018) 151–164, http://dx.doi.org/10.1007/s12293-016-0212-3, URL: http://link.springer.com/10.1007/s12293-016-0212-3.

[70] R. Salgotra, U. Singh, The naked mole-rat algorithm, Neural Comput. Appl. 31 (12) (2019) 8837–8857, http://dx.doi.org/10.1007/s00521-019-04464-7, URL: http://link.springer.com/10.1007/s00521-019-04464-7.

[71] M. Ghasemi, E. Akbari, A. Rahimnejad, S.E. Razavi, S. Ghavidel, L. Li, Phasor particle swarm optimization: a simple and efficient variant of PSO, Soft Comput. 23 (19) (2019) 9701–9718, http://dx.doi.org/10.1007/s00500-018-3536-8, URL: http://link.springer.com/10.1007/s00500-018-3536-8.

[72] M. Ghasemi, J. Aghaei, M. Hadipour, New self-organising hierarchical PSO with jumping time-varying acceleration coefficients, Electron. Lett. 53 (20) (2017) 1360–1362, http://dx.doi.org/10.1049/el.2017.2112, URL: https://onlinelibrary.wiley.com/doi/10.1049/el.2017.2112.

[73] B. Liu, L. Wang, Y.-H. Jin, F. Tang, D.-X. Huang, Improved particle swarm optimization combined with chaos, Chaos Solitons Fractals 25 (5) (2005) 1261–1271, http://dx.doi.org/10.1016/j.chaos.2004.11.095, URL: https://linkinghub.elsevier.com/retrieve/pii/S0960077905000330.

[74] J. Liang, A. Qin, P. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295, http://dx.doi.org/10.1109/TEVC.2005.857610, URL: http://ieeexplore.ieee.org/document/1637688/.

[75] H. Yapici, N. Cetinkaya, A new meta-heuristic optimizer: Pathfinder algorithm, Appl. Soft Comput. 78 (2019) 545–568, http://dx.doi.org/10.1016/j.asoc.2019.03.012, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494619301309.

[76] S. Shadravan, H. Naji, V. Bardsiri, The Sailfish Optimizer: A novel nature-inspired metaheuristic algorithm for solving constrained engineering optimization problems, Eng. Appl. Artif. Intell. 80 (2019) 20–34, http://dx.doi.org/10.1016/j.engappai.2019.01.001, URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197619300016.

[77] L.-L. Li, Q. Shen, M.-L. Tseng, S. Luo, Power system hybrid dynamic economic emission dispatch with wind energy based on improved sailfish algorithm, J. Clean. Prod. 316 (2021) 128318, http://dx.doi.org/10.1016/j.jclepro.2021.128318, URL: https://linkinghub.elsevier.com/retrieve/pii/S0959652621025336.

[78] S. Mirjalili, A.H. Gandomi, S.Z. Mirjalili, S. Saremi, H. Faris, S.M. Mirjalili, Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems, Adv. Eng. Softw. 114 (2017) 163–191, http://dx.doi.org/10.1016/j.advengsoft.2017.07.002, URL: https://linkinghub.elsevier.com/retrieve/pii/S0965997816307736.

[79] A. Seyyedabbasi, F. Kiani, Sand Cat swarm optimization: a nature-inspired algorithm to solve global optimization problems, Eng. Comput. (2022) 1–25, http://dx.doi.org/10.1007/s00366-022-01604-x.

[80] R. Masadeh, B. A., A. Sharieh, Sea Lion Optimization Algorithm, Int. J. Adv. Comput. Sci. Appl. 10 (5) (2019) http://dx.doi.org/10.14569/IJACSA.2019.0100548, URL: http://thesai.org/Publications/ViewPaper?Volume=10&Issue=5&Code=IJACSA&SerialNo=48.

[81] R. Masadeh, N. Alsharman, A. Sharieh, B.A. Mahafzah, A. Abdulrahman, Task scheduling on cloud computing based on sea lion optimization algorithm, Int. J. Web Inf. Syst. 17 (2) (2021) 99–116, http://dx.doi.org/10.1108/IJWIS-11-2020-0071, URL: https://www.emerald.com/insight/content/doi/10.1108/IJWIS-11-2020-0071/full/html.

[82] B.M. Nguyen, T. Tran, T. Nguyen, G. Nguyen, An improved sea lion optimization for workload elasticity prediction with neural networks, Int. J. Comput. Intell. Syst. 15 (1) (2022) 1–26, http://dx.doi.org/10.1007/s44196-022-00156-8.

[83] J.J. Yu, V.O. Li, A social spider algorithm for global optimization, Appl. Soft Comput. 30 (2015) 614–627, http://dx.doi.org/10.1016/j.asoc.2015.02.014, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494615001052.

[84] A. Luque-Chang, E. Cuevas, F. Fausto, D. Zaldívar, M. Pérez, Social spider optimization algorithm: Modifications, applications, and perspectives, Math. Probl. Eng. 2018 (2018) e6843923, http://dx.doi.org/10.1155/2018/6843923, URL: https://www.hindawi.com/journals/mpe/2018/6843923/.

[85] J. Xue, B. Shen, A novel swarm intelligence optimization approach: sparrow search algorithm, Syst. Sci. Control Eng. 8 (1) (2020) 22–34, http://dx.doi.org/10.1080/21642583.2019.1708830, URL: https://www.tandfonline.com/doi/full/10.1080/21642583.2019.1708830.

[86] G. Dhiman, V. Kumar, Spotted hyena optimizer: A novel bio-inspired based metaheuristic technique for engineering applications, Adv. Eng. Softw. 114 (2017) 48–70, http://dx.doi.org/10.1016/j.advengsoft.2017.05.014, URL: https://linkinghub.elsevier.com/retrieve/pii/S0965997816305567.

[87] M. Bakhshipour, M. Jabbari Ghadi, F. Namdari, Swarm robotics search & rescue: A novel artificial intelligence-inspired optimization approach, Appl. Soft Comput. 57 (2017) 708–726, http://dx.doi.org/10.1016/j.asoc.2017.02.028, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494617301072.

[88] L. Xie, T. Han, H. Zhou, Z.-R. Zhang, B. Han, A. Tang, Tuna swarm optimization: a novel swarm-based metaheuristic algorithm for global optimization, Comput. Intell. Neurosci. 2021 (2021) http://dx.doi.org/10.1155/2021/9210050.

[89] C. Tang, W. Sun, W. Wu, M. Xue, A hybrid improved whale optimization algorithm, in: 2019 IEEE 15th International Conference on Control and Automation, ICCA, IEEE, Edinburgh, United Kingdom, 2019, pp. 362–367, http://dx.doi.org/10.1109/ICCA.2019.8900003, URL: https://ieeexplore.ieee.org/document/8900003/.

[90] F.A. Hashim, K. Hussain, E.H. Houssein, M.S. Mabrouk, W. Al-Atabany, Archimedes optimization algorithm: a new metaheuristic algorithm for solving optimization problems, Appl. Intell. 51 (3) (2021) 1531–1551, http://dx.doi.org/10.1007/s10489-020-01893-z, URL: https://link.springer.com/10.1007/s10489-020-01893-z.

[91] W. Zhao, L. Wang, Z. Zhang, Atom search optimization and its application to solve a hydrogeologic parameter estimation problem, Knowl.-Based Syst. 163 (2019) 283–304, http://dx.doi.org/10.1016/j.knosys.2018.08.030, URL: https://linkinghub.elsevier.com/retrieve/pii/S0950705118304271.

[92] H. Abedinpourshotorban, S. Mariyam Shamsuddin, Z. Beheshti, D.N. Jawawi, Electromagnetic field optimization: A physics-inspired metaheuristic optimization algorithm, Swarm Evol. Comput. 26 (2016) 8–22, http://dx.doi.org/10.1016/j.swevo.2015.07.002, URL: https://linkinghub.elsevier.com/retrieve/pii/S2210650215000528.

[93] A. Faramarzi, M. Heidarinejad, B. Stephens, S. Mirjalili, Equilibrium optimizer: A novel optimization algorithm, Knowl.-Based Syst. 191 (2020) 105190, http://dx.doi.org/10.1016/j.knosys.2019.105190, URL: https://linkinghub.elsevier.com/retrieve/pii/S0950705119305295.

[94] S. Gupta, K. Deep, S. Mirjalili, An efficient equilibrium optimizer with mutation strategy for numerical optimization, Appl. Soft Comput. 96 (2020) 106542, http://dx.doi.org/10.1016/j.asoc.2020.106542, URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494620304816.

[95] A. Wunnava, M.K. Naik, R. Panda, B. Jena, A. Abraham, A novel interdependence based multilevel thresholding technique using adaptive equilibrium optimizer, Eng. Appl. Artif. Intell. 94 (2020) 103836, http://dx.doi.org/10.1016/j.engappai.2020.103836, URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197620302037.

[96] F.A. Hashim, E.H. Houssein, M.S. Mabrouk, W. Al-Atabany, S. Mirjalili, Henry gas solubility optimization: A novel physics-based algorithm, Future Gener. Comput. Syst. 101 (2019) 646–667, http://dx.doi.org/10.1016/j.future.2019.07.015, URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X19306557.

[97] S. Mirjalili, S.M. Mirjalili, A. Hatamlou, Multi-Verse Optimizer: a nature-inspired algorithm for global optimization, Neural Comput. Appl. 27 (2) (2016) 495–513, http://dx.doi.org/10.1007/s00521-015-1870-7, URL: https://doi.org/10.1007/s00521-015-1870-7.

[98] Z. Wei, C. Huang, X. Wang, T. Han, Y. Li, Nuclear reaction optimization: A novel and powerful physics-based algorithm for global optimization, IEEE Access 7 (2019) 66084–66109, http://dx.doi.org/10.1109/ACCESS.2019.2918406.

[99] A. Kaveh, Tug of war optimization, in: Advances in Metaheuristic Algorithms for Optimal Design of Structures, Springer International Publishing, Cham, 2017, pp. 451–487, http://dx.doi.org/10.1007/978-3-319-46173-1_15, URL: http://link.springer.com/10.1007/978-3-319-46173-1_15.

[100] A. Kaveh, P. Almasi, A. Khodagholi, Optimum design of castellated beams using four recently developed meta-heuristic algorithms, Iran. J. Sci. Technol. Trans. Civil Eng. (2022) http://dx.doi.org/10.1007/s40996-022-00884-z, URL: https://link.springer.com/10.1007/s40996-022-00884-z.

[101] T. Nguyen, B. Hoang, G. Nguyen, B.M. Nguyen, A new workload prediction model using extreme learning machine and enhanced tug of war optimization, Procedia Comput. Sci. 170 (2020) 362–369, http://dx.doi.org/10.1016/j.procs.2020.03.063, URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050920305007.

[102] Z. Bayraktar, M. Komurcu, J.A. Bossard, D.H. Werner, The wind driven optimization technique and its application in electromagnetics, IEEE Trans. Antennas and Propagation 61 (5) (2013) 2745–2757, http://dx.doi.org/10.1109/TAP.2013.2238654, URL: http://ieeexplore.ieee.org/document/6407788/.

[103] T. Rahkar Farshi, Battle royale optimization algorithm, Neural Comput. Appl. 33 (4) (2021) 1139–1157, http://dx.doi.org/10.1007/s00521-020-05004-4.

[104] Y. Shi, Brain storm optimization algorithm, in: International Conference in Swarm Intelligence, Springer, 2011, pp. 303–309, http://dx.doi.org/10.1007/978-3-642-21515-5_36.

[105] M. El-Abd, Global-best brain storm optimization algorithm, Swarm Evol. Comput. 37 (2017) 27–44, http://dx.doi.org/10.1016/j.swevo.2017.05.001, URL: https://linkinghub.elsevier.com/retrieve/pii/S2210650216301766.

[106] M.A. Al-Betar, Z.A.A. Alyasseri, M.A. Awadallah, I. Abu Doush, Coronavirus herd immunity optimizer (CHIO), Neural Comput. Appl. 33 (10) (2021) 5011–5042, http://dx.doi.org/10.1007/s00521-020-05296-6.

[107] B. Chen, L. Zhao, J.H. Lu, Wind power forecast using RBF network and culture algorithm, in: 2009 International Conference on Sustainable Power Generation and Supply, IEEE, 2009, pp. 1–6, http://dx.doi.org/10.1109/SUPERGEN.2009.5348174.

[108] J.-S. Chou, N.-M. Nguyen, FBI inspired meta-optimization, Appl. Soft Comput. 93 (2020) 106339, http://dx.doi.org/10.1016/j.asoc.2020.106339.

[109] A. Fathy, H. Rezk, T.M. Alanazi, Recent approach of forensic-based investigation algorithm for optimizing fractional order PID-based MPPT with proton exchange membrane fuel cell, IEEE Access 9 (2021) 18974–18992, http://dx.doi.org/10.1109/ACCESS.2021.3054552, URL: https://ieeexplore.ieee.org/document/9335569/.

[110] A.W. Mohamed, A.A. Hadi, A.K. Mohamed, Gaining-sharing knowledge based algorithm for solving optimization problems: a novel nature-inspired algorithm, Int. J. Mach. Learn. Cybern. 11 (7) (2020) 1501–1529, http://dx.doi.org/10.1007/s13042-019-01053-x.

[111] A.W. Mohamed, A.A. Hadi, A.K. Mohamed, N.H. Awad, Evaluating the performance of adaptive GainingSharing knowledge based algorithm on CEC 2020 benchmark problems, in: 2020 IEEE Congress on Evolutionary Computation, CEC, IEEE, Glasgow, United Kingdom, 2020, pp. 1–8, http://dx.doi.org/10.1109/CEC48606.2020.9185901, URL: https://ieeexplore.ieee.org/document/9185901/.

[112] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, in: 2007 IEEE Congress on Evolutionary Computation, Ieee, 2007, pp. 4661–4667, http://dx.doi.org/10.1109/CEC.2007.4425083.

[113] A. Khatri, A. Gaba, K. Rana, V. Kumar, A novel life choice-based optimizer, Soft Comput. 24 (12) (2020) 9121–9141, http://dx.doi.org/10.1007/s00500-019-04443-z.

[114] J. Zhang, M. Xiao, L. Gao, Q. Pan, Queuing search algorithm: A novel metaheuristic algorithm for solving engineering optimization problems, Appl. Math. Model. 63 (2018) 464–490, http://dx.doi.org/10.1016/j.apm.2018.06.036.

[115] X. Zheng, H. Nguyen, A novel artificial intelligent model for predicting water treatment efficiency of various biochar systems based on artificial neural network and queuing search algorithm, Chemosphere 287 (2022) 132251, http://dx.doi.org/10.1016/j.chemosphere.2021.132251.

[116] H. Abderazek, F. Hamza, A.R. Yildiz, L. Gao, S.M. Sait, A comparative analysis of the queuing search algorithm, the sine-cosine algorithm, the ant lion algorithm to determine the optimal weight design problem of a spur gear drive system, Mater. Test. 63 (5) (2021) 442–447, http://dx.doi.org/10.1515/mt-2020-0075.

[117] B.M. Nguyen, B. Hoang, T. Nguyen, G. Nguyen, nQSV-Net: a novel queuing search variant for global space search and workload modeling, J. Ambient Intell. Humaniz. Comput. 12 (1) (2021) 27–46, http://dx.doi.org/10.1007/s12652-020-02849-4.

[118] A. Shabani, B. Asgarian, S.A. Gharebaghi, M.A. Salido, A. Giret, A new optimization algorithm based on search and rescue operations, Math. Probl. Eng. 2019 (2019) 1–23, http://dx.doi.org/10.1155/2019/2482543, URL: https://www.hindawi.com/journals/mpe/2019/2482543/.

[119] A. Tharwat, T. Gabel, Parameters optimization of support vector machines for imbalanced data using social ski driver algorithm, Neural Comput. Appl. 32 (11) (2020) 6925–6938, http://dx.doi.org/10.1007/s00521-019-04159-z, URL: http://link.springer.com/10.1007/s00521-019-04159-z.

[120] B. Das, V. Mukherjee, D. Das, Student psychology based optimization algorithm: A new population based optimization algorithm for solving optimization problems, Adv. Eng. Softw. 146 (2020) 102804, http://dx.doi.org/10.1016/j.advengsoft.2020.102804.

[121] R. Rao, V. Savsani, D. Vakharia, Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems, Comput. Aided Des. 43 (3) (2011) 303–315, http://dx.doi.org/10.1016/j.cad.2010.12.015, URL: https://linkinghub.elsevier.com/retrieve/pii/S0010448510002484.

[122] R.V. Rao, V. Patel, An elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problems, Int. J. Ind. Eng. Comput. 3 (4) (2012) 535–560, http://dx.doi.org/10.5267/j.ijiec.2012.03.007, URL: http://www.growingscience.com/ijiec/Vol3/IJIEC_2012_37.pdf.

[123] R.V. Rao, V. Patel, An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems, Sci. Iran. (2012) S1026309812002672, http://dx.doi.org/10.1016/j.scient.2012.12.005, URL: https://linkinghub.elsevier.com/retrieve/pii/S1026309812002672.

[124] M.H. Sulaiman, Z. Mustaffa, M.M. Saari, H. Daniyal, Barnacles mating optimizer: a new bio-inspired algorithm for solving engineering optimization problems, Eng. Appl. Artif. Intell. 87 (2020) 103330, http://dx.doi.org/10.1016/j.engappai.2019.103330.

[125] D. Simon, Biogeography-based optimization, IEEE Trans. Evol. Comput. 12 (6) (2008) 702–713, http://dx.doi.org/10.1109/TEVC.2008.919004.

[126] G.-G. Wang, S. Deb, L.D.S. Coelho, Earthworm optimisation algorithm: a bio-inspired metaheuristic algorithm for global optimisation problems, Int. J. Bio-Inspired Comput. 12 (1) (2018) 1–22, http://dx.doi.org/10.1504/IJBIC.2015.10004283.

[127] A.R. Mehrabian, C. Lucas, A novel numerical optimization algorithm inspired from weed colonization, Ecol. Inform. 1 (4) (2006) 355–366, http://dx.doi.org/10.1016/j.ecoinf.2006.07.003.

[128] S.H.S. Moosavi, V.K. Bardsiri, Satin bowerbird optimizer: A new optimization algorithm to optimize ANFIS for software development effort estimation, Eng. Appl. Artif. Intell. 60 (2017) 1–15, http://dx.doi.org/10.1016/j.engappai.2017.01.006.

[129] G. Dhiman, V. Kumar, Seagull optimization algorithm: Theory and its applications for large-scale industrial engineering problems, Knowl.-Based Syst. 165 (2019) 169–196, http://dx.doi.org/10.1016/j.knosys.2018.11.024.

[130] S. Li, H. Chen, M. Wang, A.A. Heidari, S. Mirjalili, Slime mould algorithm: A new method for stochastic optimization, Future Gener. Comput. Syst. 111 (2020) 300–323, http://dx.doi.org/10.1016/j.future.2020.03.055.

[131] M.-Y. Cheng, D. Prayogo, Symbiotic organisms search: a new metaheuristic optimization algorithm, Comput. Struct. 139 (2014) 98–112, http://dx.doi.org/10.1016/j.compstruc.2014.03.007.

[132] S. Kaur, L.K. Awasthi, A. Sangal, G. Dhiman, Tunicate Swarm Algorithm: A new bio-inspired based metaheuristic paradigm for global optimization, Eng. Appl. Artif. Intell. 90 (2020) 103541, http://dx.doi.org/10.1016/j.engappai.2020.103541.

[133] M.D. Li, H. Zhao, X.W. Weng, T. Han, A novel nature-inspired algorithm for optimization: Virus colony search, Adv. Eng. Softw. 92 (2016) 65–88, http://dx.doi.org/10.1016/j.advengsoft.2015.11.004.

[134] D. Amali, M. Dinakaran, Wildebeest herd optimization: a new global optimization algorithm inspired by wildebeest herding behaviour, J. Intell. Fuzzy Systems 37 (6) (2019) 8063–8076, http://dx.doi.org/10.3233/JIFS-190495.

[135] C. Villaseñor, N. Arana-Daniel, A.Y. Alanis, C. López-Franco, E.A. Hernandez-Vargas, Germinal center optimization algorithm, Int. J. Comput. Intell. Syst. 12 (1) (2018) 13, http://dx.doi.org/10.2991/ijcis.2018.25905179, URL: https://www.atlantis-press.com/article/25905179.

[136] H. Eskandar, A. Sadollah, A. Bahreininejad, M. Hamdi, Water cycle algorithm – A novel metaheuristic optimization method for solving constrained engineering optimization problems, Comput. Struct. 110–111 (2012) 151–166, http://dx.doi.org/10.1016/j.compstruc.2012.07.010, URL: https://linkinghub.elsevier.com/retrieve/pii/S0045794912001770.

[137] W. Zhao, L. Wang, Z. Zhang, Artificial ecosystem-based optimization: a novel nature-inspired meta-heuristic algorithm, Neural Comput. Appl. 32 (13) (2020) 9383–9425, http://dx.doi.org/10.1007/s00521-019-04452-x, URL: http://link.springer.com/10.1007/s00521-019-04452-x.

[138] A. Eid, S. Kamel, A. Korashy, T. Khurshaid, An enhanced artificial ecosystem-based optimization for optimal allocation of multiple distributed generations, IEEE Access 8 (2020) 178493–178513, http://dx.doi.org/10.1109/ACCESS.2020.3027654, URL: https://ieeexplore.ieee.org/document/9208695/.

[139] A. S. Menesy, H.M. Sultan, A. Korashy, F.A. Banakhr, M. G. Ashmawy, S. Kamel, Effective parameter extraction of different polymer electrolyte membrane fuel cell stack models using a modified artificial ecosystem optimization algorithm, IEEE Access 8 (2020) 31892–31909, http://dx.doi.org/10.1109/ACCESS.2020.2973351, URL: https://ieeexplore.ieee.org/document/8995515/.

[140] R.M. Rizk-Allah, A.A. El-Fergany, Artificial ecosystem optimizer for parameters identification of proton exchange membrane fuel cells model, Int. J. Hydrogen Energy 46 (75) (2021) 37612–37627, http://dx.doi.org/10.1016/j.ijhydene.2020.06.256, URL: https://linkinghub.elsevier.com/retrieve/pii/S0360319920324472.

[141] N. Van Thieu, S.D. Barma, T. Van Lam, O. Kisi, A. Mahesha, Groundwater level modeling using Augmented Artificial Ecosystem Optimization, J. Hydrol. (2022) 129034, http://dx.doi.org/10.1016/j.jhydrol.2022.129034, URL: https://doi.org/10.1016/j.jhydrol.2022.129034.

[142] L. Abualigah, A. Diabat, S. Mirjalili, M. Abd Elaziz, A.H. Gandomi, The arithmetic optimization algorithm, Comput. Methods Appl. Mech. Engrg. 376 (2021) 113609, http://dx.doi.org/10.1016/j.cma.2020.113609, URL: https://linkinghub.elsevier.com/retrieve/pii/S0045782520307945.

[143] S. Talatahari, M. Azizi, Chaos game optimization: a novel metaheuristic algorithm, Artif. Intell. Rev. 54 (2) (2021) 917–1004, http://dx.doi.org/10.1007/s10462-020-09867-w, URL: https://link.springer.com/10.1007/s10462-020-09867-w.

[144] M.H. Qais, H.M. Hasanien, R.A. Turky, S. Alghuwainem, M. Tostado-Véliz, F. Jurado, Circle search algorithm: A geometry-based metaheuristic optimization algorithm, Mathematics 10 (10) (2022) 1626, http://dx.doi.org/10.3390/math10101626.

[145] R. Rubinstein, The cross-entropy method for combinatorial and continuous optimization, Methodol. Comput. Appl. Probab. 1 (2) (1999) 127–190, http://dx.doi.org/10.1023/A:1010091220143.

[146] I. Ahmadianfar, O. Bozorg-Haddad, X. Chu, Gradient-based optimizer: A new metaheuristic optimization algorithm, Inform. Sci. 540 (2020) 131–159, http://dx.doi.org/10.1016/j.ins.2020.06.037, URL: https://linkinghub.elsevier.com/retrieve/pii/S0020025520306241.

[147] A. Prügel-Bennett, When a genetic algorithm outperforms hill-climbing, Theoret. Comput. Sci. 320 (1) (2004) 135–153, http://dx.doi.org/10.1016/j.tcs.2004.03.038, URL: https://linkinghub.elsevier.com/retrieve/pii/S0304397504001999.

[148] M. Shaqfa, K. Beyer, Pareto-like sequential sampling heuristic for global optimisation, Soft Comput. 25 (14) (2021) 9077–9096, http://dx.doi.org/10.1007/s00500-021-05853-8, URL: https://link.springer.com/10.1007/s00500-021-05853-8.

[149] I. Ahmadianfar, A.A. Heidari, A.H. Gandomi, X. Chu, H. Chen, RUN beyond the metaphor: An efficient optimization algorithm based on runge kutta method, Expert Syst. Appl. 181 (2021) 115079, http://dx.doi.org/10.1016/j.eswa.2021.115079.

[150] S. Mirjalili, SCA: A Sine cosine algorithm for solving optimization problems, Knowl.-Based Syst. 96 (2016) 120–133, http://dx.doi.org/10.1016/j.knosys.2015.12.022, URL: https://linkinghub.elsevier.com/retrieve/pii/S0950705115005043.

[151] I. Ahmadianfar, A.A. Heidari, S. Noshadian, H. Chen, A.H. Gandomi, INFO: An efficient optimization algorithm based on weighted mean of vectors, Expert Syst. Appl. 195 (2022) 116516, http://dx.doi.org/10.1016/j.eswa.2022.116516.

[152] K. Hussain, M.N.M. Salleh, S. Cheng, Y. Shi, On the exploration and exploitation in popular swarm-based metaheuristic algorithms, Neural Comput. Appl. 31 (11) (2019) 7665–7683, http://dx.doi.org/10.1007/s00521-018-3592-0, URL: http://link.springer.com/10.1007/s00521-018-3592-0.

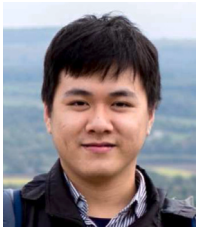
[153] X.-S. Yang, Nature-Inspired Optimization Algorithms, first ed., Elsevier, Amsterdam, Boston, 2014, http://dx.doi.org/10.1016/C2013-0-01368-0.

[154] P.-A. Simionescu, D.G. Beale, New concepts in graphic visualization of objective functions, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 36223, 2002, pp. 891–897, http://dx.doi.org/10.1115/DETC2002/DAC-34129.

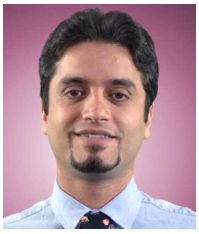
[155] W.J. Lim, A.B. Jambek, S.C. Neoh, Kursawe and ZDT functions optimization using hybrid micro genetic algorithm (HMGA), Soft Comput. 19 (12) (2015) 3571–3580, http://dx.doi.org/10.1007/s00500-015-1767-5.

[156] G. Wu, R. Mallipeddi, P.N. Suganthan, Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization, National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report, 2017.

[157] W.H. Wolberg, W.N. Street, O.L. Mangasarian, Image analysis and machine learning applied to breast cancer diagnosis and prognosis, Anal. Quant. Cytol. Histol. 17 (2) (1995) 77–87.



Nguyen Van Thieu is currently a lecturer and researcher at Phenikaa University, Vietnam. He received his Bachelor's and Master's degrees from the School of Information and Communication Technology at Hanoi University of Science and Technology (HUST) through the highly competitive ICT Talented Engineering program. Since 2018, he has made significant contributions to the field of computer science as the first author of many papers in international conferences and journals. His research topics are focused on nature-inspired computing, machine learning, deep learning, neural networks, and meta-heuristic algorithms. Specifically, his research interests encompass natural language processing, cloud resource management, log template generation for large-scale systems, optimization in engineering applications, and time-series forecasting in hydrology.



Seyedali Mirjalili is a Professor and the director of the Center for Artificial Intelligence Research and Optimization at Torrens University Australia. He has gained international recognition for his contributions to nature-inspired artificial intelligence techniques, with over 500 published works, 70,000 citations, and an H-index of 85. He was included on the list of the top 1% of highly-cited researchers in 2019, and Web of Science named him one of the most influential researchers in the world. In 2022 and 2023, The Australian newspaper recognized him as a global leader in Artificial Intelligence and a national leader in the Evolutionary Computation and Fuzzy Systems fields. Additionally, he serves as a senior member of IEEE and holds editorial positions at several top AI journals, including Engineering Applications of Artificial Intelligence, Applied Soft Computing, Neurocomputing, Advances in Engineering Software, Computers in Biology and Medicine, Applied Intelligence, and Decision Analytics.