PlatEMO

Evolutionary Multi-Objective Optimization Platform

User Manual 4.2

BIMK Group

May 28, 2023

Thank you very much for using PlatEMO. The copyright of PlatEMO belongs to the BIMK Group of Anhui University, China. This platform is only for research and educational purposes. The codes were implemented based on our understanding of the algorithms published in literatures. You should not rely upon the material or information provided by the platform as a basis for making any business, legal or any other decisions. We assume no responsibilities for any consequences of your using any codes in the platform. All publications using the platform should acknowledge the use of "PlatEMO" and cite one of the following two papers:

[1] Ye Tian, Ran Cheng, Xingyi Zhang, and Yaochu Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum]," IEEE Computational Intelligence Magazine, 2017, 12(4): 73-87.

[2] Ye Tian, Weijian Zhu, Xingyi Zhang, and Yaochu Jin, "A practical tutorial on solving optimization problems via PlatEMO," Neurocomputing, 2023, 518: 190-205.

If you have any comment or suggestion to PlatEMO, please send it to *field910921@gmail.com* (*Dr. Ye Tian*). If you want to add your code to PlatEMO, please send the ready-to-use code and the relevant literature to *field910921@gmail.com* as well. You can obtain the newest version of PlatEMO from GitHub.

Contents

I. Quick Start

Requirement: MATLAB R2018a or higher (PlatEMO without GUI) or
MATLAB R2020b or higher (PlatEMO with GUI) with
Parallel Computing Toolbox and
Statistics and Machine Learning Toolbox

PlatEMO is an open-source platform for solving optimization problems, whose input is an optimization problem and output is the found optimal solutions. An optimization problem is defined as

$$\begin{aligned}\min_{\mathbf{x}}\ & \mathbf{f}(\mathbf{x}) = \left(f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})\right)\\ \text{s.t.}\ & \mathbf{x} = (x_1, x_2, \dots x_D) \in \Omega\\ & g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_K(\mathbf{x}) \le 0\end{aligned}$$

where **x** denotes a **solution** or **decision vector** for the problem, which consists of D **decision variables** x_i, and each decision variable can be a real number, integer, binary number, or others. Ω denotes the **search space** of the problems, which consists of the **lower bound** $l_1, l_2, \dots l_D$ and the **upper bound** $u_1, u_2, \dots u_D$, i.e., each decision variable should always satisfy that $l_i \le x_i \le u_i$. $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})$ denote the M **objective values** of the solution, and $g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_K(\mathbf{x})$ denote the K **constraint violations** of the solution.

To define an optimization problem, users should input at least the following contents:

- The encoding scheme of each decision variable (real, integer, binary, etc.);
- The lower bound $l_1, l_2, \dots l_D$ and the upper bound $u_1, u_2, \dots u_D$;
- At least one objective function $f_1(\mathbf{x})$.

To define an optimization problem more precisely, users can also input the following contents:

- Multiple objective functions $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})$;
- Multiple constraint functions $g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_K(\mathbf{x})$;
- Function for initializing solutions;
- Function for repairing invalid solutions;
- Function for evaluating solutions;
- Gradient functions of objectives $f_1'(\mathbf{x}), f_2'(\mathbf{x}), \dots, f_M'(\mathbf{x})$;
- Gradient functions of constraints $g_1'(\mathbf{x}), g_2'(\mathbf{x}), \dots, g_K'(\mathbf{x})$;
- Data used in the calculation of all functions (an arbitrary constant).

The above functions are MATLAB functions rather than mathematical functions, which

should have specified inputs and outputs but need not have explicit mathematical expressions. Moreover, users can define the settings of optimization algorithms, to achieve the improvement of optimization performance via selecting suitable algorithms and parameter settings.

In MATLAB, users can call the main file `platemo.m` in the following three ways:

1) Calling the main function with parameters:

```
platemo('problem',@SOP_F1,'algorithm',@GA);
```

Then the specified benchmark problem will be solved by the specified algorithm with specified parameter settings, where the result can be displayed, saved, or returned (see *Solving Benchmark Problems* for details).

2) Calling the main function with parameters:

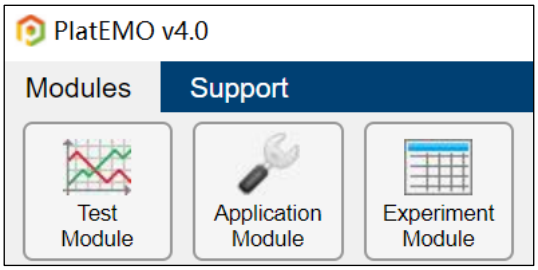
```
f1 = @(x)sum(x);
f2 = @(x)1-sum(x);
platemo('objFcn',f1,'conFcn',f2,'algorithm',@GA);
```

Then the user-defined problem will be solved by the specified algorithm with specified parameter settings (see *Solving User-Defined Problems* for details).

3) Calling the main function without parameter:

```
platemo();
```

Then a GUI with three modules will be displayed, where the test module is used to visually investigate the performance of an algorithm on a benchmark problem (see *Functions of Test Module* for details), the application module is used to solve user-defined problems (see *Functions of Application Module* for details), and the experiment module is used to statistically analyze the performance of multiple algorithms on multiple benchmark problems (see *Functions of Experiment Module* for details).



II. Using PlatEMO without GUI

A. Solving Benchmark Problems

Users can use PlatEMO without GUI by calling the main function `platemo()` with parameters like

```
platemo('Name1',Value1,'Name2',Value2,'Name3',Value3,…);
```

where all the acceptable names and values are

Name	Data type	Default value	Description
`'algorithm'`	Function handle or cell	`dependent`	Class of algorithm
`'problem'`	Function handle or cell	`dependent`	Class of problem
`'N'`	Positive integer	`100`	Population size
`'M'`	Positive integer	`dependent`	Number of objectives
`'D'`	Positive integer	`dependent`	Number of variables
`'maxFE'`	Positive integer	`10000`	Maximum number of function evaluations
`'maxRuntime'`	Positive number	`inf`	Maximum runtime
`'save'`	Integer	`-10`	Number of saved populations
`'outputFcn'`	Function handle	`@DefaultOutput`	Function called before each iteration Input 1: Class of algorithm Input 2: Class of problem Output: None

- `'algorithm'` denotes the algorithm to be run, whose value should be the function handle of an algorithm, such as `@GA`. The value can also be a cell like `{@GA,p1,p2,…}`, where `p1,p2,…` specify the parameter values of the algorithm. For example, the following code solves the default problem via the algorithm `@GA` with specified parameters:

```
platemo('algorithm',{@GA,1,30,1,30});
```

- `'problem'` denotes the benchmark problem to solve, whose value should be the function handle of a benchmark problem, such as `@SOP_F1`. The value can also be a cell like `{@SOP_F1,p1,p2,…}`, where `p1,p2,…` specify the parameter values of the benchmark problem. For example, the following code solves the problem

`@WFG1` with specified parameters via the default algorithm:

```
platemo('problem',{@WFG1,20});
```

- `'N'` denotes the population size of the algorithm, which usually equals the number of solutions in the final population. For example, the following code solves the problem `@SOP_F1` via the algorithm `@GA` with a population size of 50:

```
platemo('algorithm',@GA,'problem',@SOP_F1,'N',50);
```

- `'M'` denotes the number of objectives of the benchmark problem, which is valid for some multi-objective benchmark problems. For example, the following code solves the problem `@SOP_F1` via the algorithm `@GA` with a population size of 50:

```
platemo('algorithm',@GA,'problem',@SOP_F1,'N',50);
```

- `'D'` denotes the number of decision variables of the benchmark problem, which is valid for some benchmark problems. For example, the following code solves the problem `@SOP_F1` with 100 variables via the algorithm `@GA`:

```
platemo('algorithm',@GA,'problem',@SOP_F1,'D',100);
```

- `'maxFE'` denotes the maximum number of available function evaluations, which usually equals the product of population size and number of generations. For example, the following code sets the maximum number of function evaluations to 20000 for the algorithm:

```
platemo('algorithm',@GA,'problem',@SOP_F1,'maxFE',20000);
```

- `'maxRuntime'` denotes the maximum runtime (in second). When `'maxRuntime'` equals its default value `inf`, the algorithm will terminate after `'maxFE'` function evaluations; otherwise, the algorithm will terminate after `'maxRuntime'` seconds. For example, the following code sets the maximum runtime to 10 seconds for the algorithm:

```
platemo('algorithm',@GA,'problem',@SOP_F1,'maxRuntime',10);
```

- `'save'` denotes the number of saved populations, where the populations are saved to a file if the value is positive and displayed in a figure if the value is negative (see *Collecting the Results* for details).
- `'outputFcn'` denotes the function called before each iteration of the algorithm. An output function has two inputs and no output, where the first input is the current `ALGORITHM` object and the second input is the current `PROBLEM` object. The default `'outputFcn'` saves or displays the populations according the value of `'save'`.

Note that users need not specify all the parameters as each of them has a default value.

B. Solving User-Defined Problems

When the parameter `'problem'` is not specified, users can define their own problem by specifying the following parameters:

Name	Data type	Default value	Description
`'objFcn'`	Function handle, matrix, or cell	`{}`	Objective functions; all the objectives are to be minimized Input: A decision vector Output: Objective value (scalar)
`'encoding'`	Scalar or row vector	`1`	Encoding scheme of each variable
`'lower'`	Scalar or row vector	`0`	Lower bound of each variable
`'upper'`	Scalar or row vector	`1`	Upper bound of each variable
`'conFcn'`	Function handle, matrix, or cell	`{}`	Constraint functions; a constraint is satisfied if and only if the constraint violation is not positive Input: A decision vector Output: Constraint violation (scalar)
`'decFcn'`	Function handle	`{}`	Function for repairing an invalid solution Input: A decision vector Output: Repaired decision vector
`'evalFcn'`	Function handle	`{}`	Function for evaluating a solution Input: A decision vector Output 1: Repaired decision vector Output 2: All objective values (vector) Output 3: All constraint violations (vector)
`'initFcn'`	Function handle	`{}`	Function for initializing a population Input: Population size Output: A matrix consisting of the decision vectors of all solutions
`'objGradFcn'`	Function handle or cell	`{}`	Gradient functions of objectives Input: A decision vector Output: Gradient (vector)
`'conGradFcn'`	Function handle or cell	`{}`	Gradient functions of constraints Input: A decision vector Output: Gradient (vector)
`'data'`	Any	`{}`	Data of the problem

- `'objFcn'` denotes the objective functions of the problem, whose value can be a function handle (a single objective), a matrix (a function is automatically fitted),

or cell (multiple objectives). An objective function has one input and one output, where the input is a decision vector and the output is the objective value. All the objectives are to be minimized. For example, the following code solves a bi-objective optimization problem with six real variables via the default algorithm:

```
f1 = @(x)x(1)+sum(x(2:end));
f2 = @(x)sqrt(1-x(1)^2)+sum(x(2:end));
platemo('objFcn',{f1,f2},'D',6);
```

where the first objective is $x_1 + \sum_{i=2}^{D} x_i$ and the second objective is $\sqrt{1-x_1^2} + \sum_{i=2}^{D} x_i$. If an objective function is a matrix, a function will be automatically fitted via Gaussian process regression, where each row of the matrix is a sample and each column of the matrix is a variable (except for the last column) or a function value (the last column). For example, the following code solves the same problem, while the objective functions are automatically fitted:

```
x = rand(50,6);
y1 = x(:,1)+sum(x(:,2:end),2);
y2 = sqrt(1-x(:,1).^2)+sum(x(:,2:end),2);
platemo('objFcn',{[x,y1],[x,y2]},'D',6);
```

- `'encoding'` denotes the encoding scheme of each variable, whose value can be a scalar or row vector, and the value of each dimension can be `1` (real number), `2` (integer), `3` (label), `4` (binary number), or `5` (permutation number). The algorithms may generate solutions via different strategies for different encoding schemes. For example, the following code specifies three real variables, two integer variables, and one binary variable:

```
f1 = @(x)x(1)+sum(x(2:end));
f2 = @(x)sqrt(1-x(1)^2)+sum(x(2:end));
platemo('objFcn',{f1,f2},'encoding',[1,1,1,2,2,4]);
```

the number of variables D is automatically set to the length of `'encoding'`.

- `'lower'` and `'upper'` denote the lower and upper bound of each variable, respectively, whose values can be scalars or row vectors, and the value of each dimension should be real. `'lower'` and `'upper'` should have the same length as `'encoding'`. For example, the following code specifies a search space $[0,1] \times [0,9]^5$:

```
f1 = @(x)x(1)+sum(x(2:end));
f2 = @(x)sqrt(1-x(1)^2)+sum(x(2:end));
platemo('objFcn',{f1,f2},'encoding',[1,1,1,2,2,4],...
'lower',0,'upper',[1,9,9,9,9,9]);
```

- `'conFcn'` denotes the constraint functions of the problem, whose value can be a function handle (a single constraint), a matrix (a function is automatically fitted), or cell (multiple constraints). A constraint function has one input and one output, where the input is a decision vector and the output is the constraint violation. A constraint is satisfied if and only if the constraint violation is not positive. For example, the following code solves a bi-objective optimization problem via the default algorithm:

```
f1 = @(x)x(1)+sum(x(2:end));
f2 = @(x)sqrt(1-x(1)^2)+sum(x(2:end));
g1 = @(x)1-sum(x(2:end));
platemo('objFcn',{f1,f2},'encoding',[1,1,1,2,2,4],...
'conFcn',g1,'lower',0,'upper',[1,9,9,9,9,9]);
```

  and adds a constraint $\sum_{i=2}^{6} x_i \geq 1$. Note that equality constraints should be converted into inequality constraints, the details of which can be found in Section 3.2 of *this paper*. If a constraint function is a matrix, a function will be automatically fitted via Gaussian process regression, where each row of the matrix is a sample and each column of the matrix is a variable (except for the last column) or a function value (the last column). For example, the following code solves the same problem, while the constraint function is automatically fitted:

```
f1 = @(x)x(1)+sum(x(2:end));
f2 = @(x)sqrt(1-x(1)^2)+sum(x(2:end));
x = rand(50,6);
y = 1-sum(x(:,2:end),2);
platemo('objFcn',{f1,f2},'encoding',[1,1,1,2,2,4],...
'conFcn',[x,y],'lower',0,'upper',[1,9,9,9,9,9]);
```

- `'decFcn'` denotes the function for repairing an invalid solution, whose value should be a function handle having one input and one output, where the input is a decision vector and the output is the repaired decision vector. For example, the following code makes x_1 always be a multiple of 0.1:

```
f1 = @(x)x(1)+sum(x(2:end));
f2 = @(x)sqrt(1-x(1)^2)+sum(x(2:end));
g1 = @(x)1-sum(x(2:end));
h = @(x)[round(x(1)/0.1)*0.1,x(2:end)];
platemo('objFcn',{f1,f2},'encoding',[1,1,1,2,2,4],...
'conFcn',g1,'decFcn',h,'lower',0,'upper',[1,9,9,9,9,9]);
```

- `'evalFcn'` denotes the function for evaluating a solution, whose value should be a function handle having one input and three output, where the input is a decision vector, the first output is the repaired decision vector, the second output is the

vector of objective values, and the third vector is the vector of constraint violations. The default `'evalFcn'` calls `'decFcn'`, `'objFcn'`, and `'conFcn'` in sequence to evaluate a solution, while the following code defines a new `'evalFcn'` to achieve solution repair, objective calculation, and constraint calculation simultaneously:

```
function [x,f,g] = Eval(x)
    x = [round(x(1)/0.1)*0.1,x(2:end)];
    x = max(0,min([1,9,9,9,9,9],x));
    f(1) = x(1)+sum(x(2:end));
    f(2) = sqrt(1-x(1)^2)+sum(x(2:end));
    g = 1-sum(x(2:end));
end
```

Then, the following codes solve the same problem by specifying only the evaluation function:

```
platemo('evalFcn',@Eval,'encoding',[1,1,1,2,2,4],...
'lower',0,'upper',[1,9,9,9,9,9]);
```

- `'initFcn'` denotes the function for initializing a population, whose value should be a function handle having one input and one output, where the input is the number of solutions in the population and the output is a matrix consisting of the decision vectors in the population. The default `'initFcn'` randomly generates solutions in the whole search space, while the following code defines a new `'initFcn'` to accelerate the convergence:

```
q = @(N)rand(N,6);
platemo('evalFcn',@Eval,'encoding',[1,1,1,2,2,4],...
'initFcn',q,'lower',0,'upper',[1,9,9,9,9,9]);
```

- `'objGradFcn'` and `'conGradFcn'` denote the gradient functions of objectives and constraints, respectively, whose values can be function handles or cells. Each gradient function should have one input and one output, where the input is a decision vector and the output is the gradient. The default gradient function estimates the gradient via finite difference, while the following code defines a new `'objGradFcn'` to accelerate the convergence and ensure the population diversity:

```
fg = @(x)[0,x(2:end)];
platemo('evalFcn',@Eval,'encoding',[1,1,1,2,2,4],...
'objGradFcn',fg,'lower',0,'upper',[1,9,9,9,9,9]);
```

Note that only a few algorithms will use gradient information.

- `'data'` denotes the data of the problem, which can be a constant of any type. If

`'data'` is specified, all the above functions should have an additional input to receive `'data'`. For example, the following code solves a rotated single-objective optimization problem:

```
d = rand(RandStream('mlfg6331_64','Seed',28),10)*2-1;
[d,~] = qr(d);
f1 = @(x,d)sum((x*d-0.5).^2);
platemo('objFcn',f1,'encoding',ones(1,10),'data',d);
```

In addition to the above way for defining a problem, a problem object can be created and solved by specified algorithm objects. For example, the following code solves the problem via the algorithm `@GA` and the algorithm `@DE`.

```
d = rand(RandStream('mlfg6331_64','Seed',28),10)*2-1;
[d,~] = qr(d);
f1 = @(x,d)sum((x*d-0.5).^2);
PRO = UserProblem('objFcn',f1,'encoding',ones(1,10),'data',d);
ALG1 = GA();
ALG2 = DE();
ALG1.Solve(PRO);
ALG2.Solve(PRO);
```

C. Collecting the Results

The generated populations can be displayed, saved, or returned after the algorithm terminates. If the main function is called like

```
[Dec,Obj,Con] = platemo(…);
```

Then the final population will be returned, where `Dec` is a matrix consisting of the decision vectors in the final population, `Obj` is a matrix consisting of the objective values in the final population, and `Con` is a matrix consisting of the constraint violations in the final population. If the main function is called like

```
platemo('save',Value,…);
```

Then the generated populations will be displayed in a figure if `Value` is negative (default), where various plots can be displayed by switching the `Data source` menu on the figure. While if `Value` is positive, the generated populations will be saved to a MAT file named as `PlatEMO\Data\alg\alg_pro_M_D_run.mat`, where `alg` is the algorithm name, `pro` is the problem name, `M` is the number of objectives, `D` is the number of variables, and `run` automatically increases from 1 until the file name does not exist. A file saves a cell `result` consisting of the generated populations and a struct `metric` consisting of the metric values. The whole optimization process of the

algorithm is divided into `Value` equal intervals, where the first column of `result` stores the number of consumed function evaluations at the last iteration of each interval, the second column of `result` stores the population at the last iteration of each interval, and `metric` stores the metric values of the stored populations. The above are achieved by the default output function `@DefaultOutput`, while users can collect the results in their own ways by specifying the value of `'outputFcn'` to the handle of a user-defined output function.

```
result =
  6×2 cell array
    {[ 1650]}    {1×50 SOLUTION}
    {[ 3300]}    {1×50 SOLUTION}
    {[ 5000]}    {1×50 SOLUTION}
    {[ 6650]}    {1×50 SOLUTION}
    {[ 8300]}    {1×50 SOLUTION}
    {[10000]}    {1×50 SOLUTION}
```

```
metric =
  struct with fields:

    runtime: 0.3317
        IGD: [6×1 double]
```

Besides, the metric values can be automatically calculated and saved in the experiment module of the GUI. To calculate the metric values manually, users should load a population, construct a `PROBLEM` object, and call its method `CalMetric`, for example,

```
% Load result before performing the following code
pro = DTLZ2();
pro.CalMetric('IGD',result{end});
```

where `'IGD'` is the name of the calculated metric (see *Metric Function* for details). In particular, IGD and HV are the most popular metrics for multi-objective optimization, whose application scopes and methods for defining reference points can be found in Section 5.3 of *this paper*.

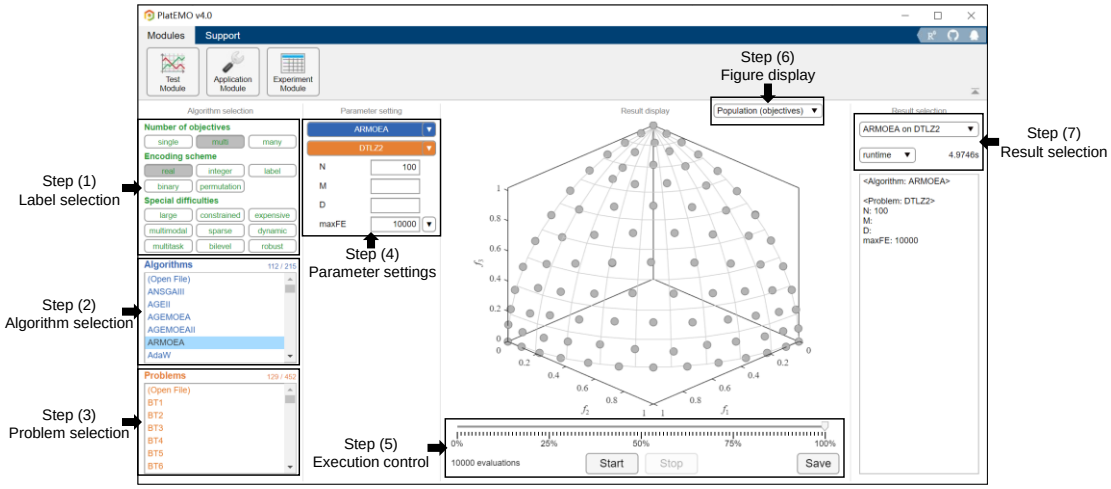
III. Using PlatEMO with GUI

A. *Functions of Test Module*

Users can use PlatEMO with GUI by calling the main function `platemo()` without parameter like

```
platemo();
```

Then the test module of the GUI will be displayed, which is used to visually investigate the performance of an algorithm on a benchmark problem.

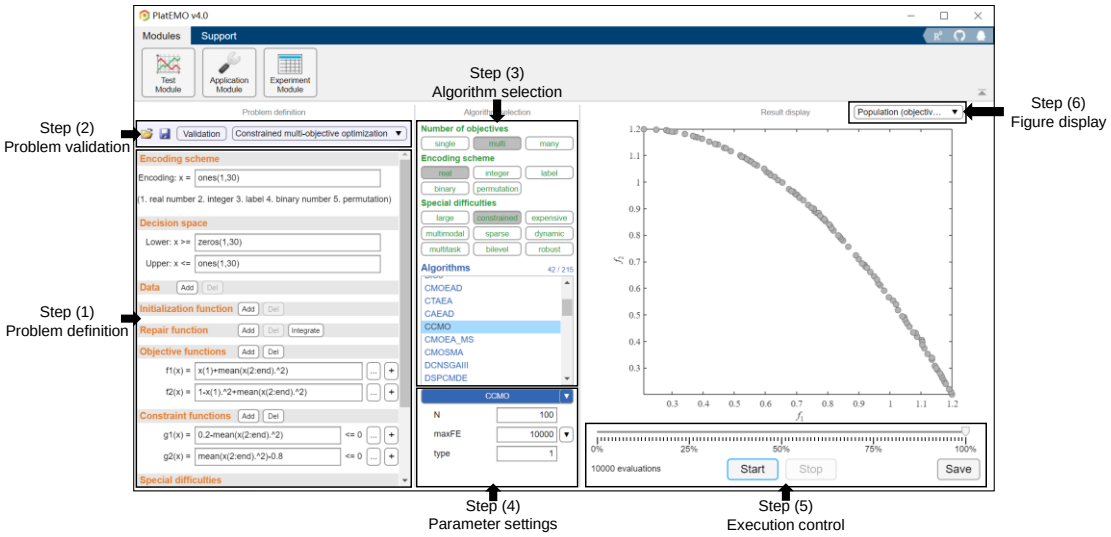


In this module, the performance investigation can be achieved by the following steps:

- Step (1) Select multiple labels to determine the type of problems (see *Labels of Algorithms, Problems, and Metrics* for details).
- Step (2) Select an algorithm from the list.
- Step (3) Select a benchmark problem from the list.
- Step (4) Set the parameters of the algorithm and benchmark problem. Different algorithms and benchmark problems may have different parameters, the details of which can be obtained by hovering over each parameter.
- Step (5) Start, pause, stop, or back off the current execution; save the current result to a file. The current result can be saved as a matrix with N rows and $D + M + K$ columns, where N denotes the number of solutions, D denotes the number of variables, M denotes the number of objectives, and K denotes the number of constraints.
- Step (6) Select a data to display, such as the objective values, variables, and metric values of the current population.
- Step (7) Select a historical result to display.

B. Functions of Application Module

Users can press the menu button to switch to the application module, which is used to solve user-defined problems.

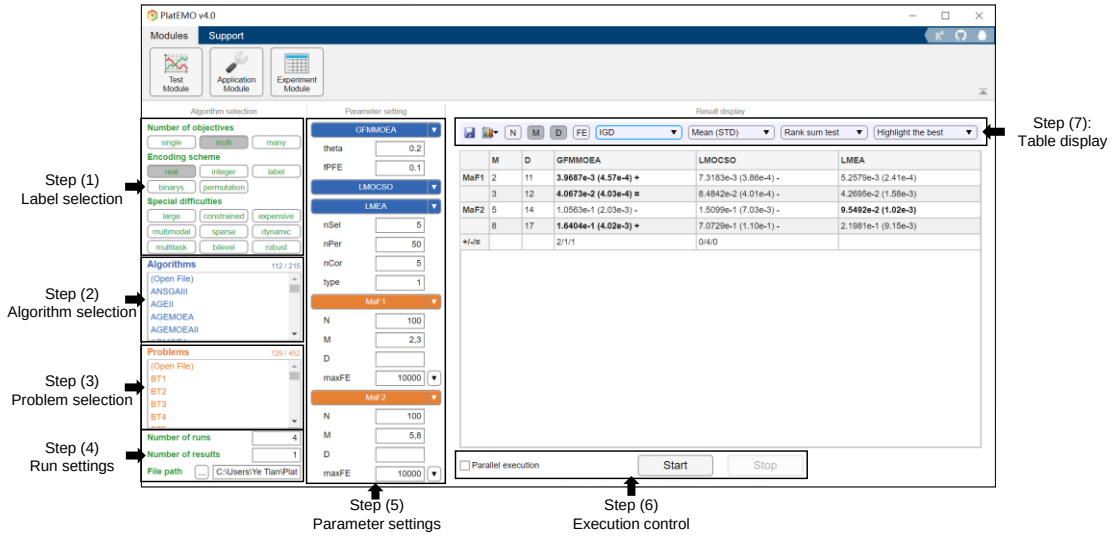


In this module, the solving of problems can be achieved by the following steps:

- Step (1) Define a problem, the contents of which are the same as those in *Solving User-Defined Problems*, where Encoding scheme corresponds to `'encoding'`, Decision space corresponds to `'lower'` and `'upper'`, Data corresponds to `'data'`, Initialization function corresponds to `'initFcn'`, Repair function corresponds to `'decFcn'`, Objective functions corresponds to `'objFcn'`, Constraint functions corresponds to `'conFcn'`, and Evaluation function corresponds to `'evalFcn'`.
- Step (2) Save or load a problem; check the validity of the problem; select a problem template. The saved problem can be opened and solved in other modules.
- Step (3) Select an algorithm from the list. The labels are automatically determined according to the problem definition (see *Labels of Algorithms, Problems, and Metrics* for details).
- Step (4) Set the parameters of the algorithm. Different algorithms may have different parameters, the details of which can be obtained by hovering over each parameter.
- Step (5) Start, pause, stop, or back off the current execution; save the current result to a file. The current result can be saved as a matrix with N rows and $D + M + K$ columns, where N denotes the number of solutions, D denotes the number of variables, M denotes the number of objectives, and K denotes the number of constraints.
- Step (6) Select a data to display, such as the objective values, variables, and metric values of the current population.

C. Functions of Experiment Module

Users can press the menu button to switch to the experiment module, which is used to statistically analyze the performance of multiple algorithms on multiple problems.



In this module, comparative experiments can be achieved by the following steps:

- Step (1) Select multiple labels to determine the type of problems (see *Labels of Algorithms, Problems, and Metrics* for details).
- Step (2) Select multiple algorithms from the list.
- Step (3) Select multiple benchmark problems from the list.
- Step (4) Set the number of repeated runs, number of saved populations in each run, and path for saving results (see *Collecting the Results* for details).
- Step (5) Set the parameters of the algorithms and benchmark problems. Different algorithms and benchmark problems may have different parameters, the details of which can be obtained by hovering over each parameter.
- Step (6) Start or stop the experiment; perform multiple runs in sequence (on a single CPU) or in parallel (on all CPUs).
- Step (7) Select a metric; select a statistical method; save the table to a file; display the results of the selected cells in a figure.

D. Labels of Algorithms, Problems, and Metrics

Each algorithm, benchmark problem, and metric should be tagged with labels by the comment in the second line of its main function. For example, in the code of `PSO.m`:

```
classdef PSO < ALGORITHM
% <single> <real/integer> <large/none> <constrained/none>
```

which indicates the types of problems the algorithm can solve. All the labels are

Label	Description
<single>	Single-objective optimization: The problem has a single objective
<multi>	Multi-objective optimization: The problem has two or three objectives
<many>	Many-objective optimization: The problem has four or more objectives
<real>	Continuous optimization: The decision variables are real numbers
<integer>	Integer optimization: The decision variables are integers
<label>	Label optimization: The decision variables are labels
<binary>	Binary optimization: The decision variables are binary numbers
<permutation>	Permutation optimization: All decision variables constitute a permutation
<large>	Large-scale optimization: The problem has 100 or more variables
<constrained>	Constrained optimization: The problem has at least one constraint
<expensive>	Expensive optimization: The objectives are computationally expensive, only a limited number of function evaluations are available
<multimodal>	Multimodal optimization: There exist multiple optimal solutions with similar objective values but considerably different decision vectors, all of which should be found
<sparse>	Sparse optimization: Most variables of the optimal solutions are zero
<dynamic>	Dynamic optimization: The objectives and constraints vary over time
<multitask>	Multitasking optimization: Optimize multiple problems simultaneously, each problem may have multiple objectives and constraints
<bilevel>	Bilevel optimization: Find the feasible and optimal solution for the upper-level problem, where a solution is feasible for the upper-level problem if and only if it is optimal for the lower-level problem
<robust>	Robust optimization: The objectives and constraints are affected by noise, the robust and optimal solutions should be found
<none>	Empty label
<min>	(for metrics only) The metric value is the smaller the better
<max>	(for metrics only) The metric value is the larger the better

An algorithm may have multiple sets of labels, where the Cartesian product between all the label sets constitutes all the types of problems that can be solved by the algorithm. If the label sets of an algorithm are <single> <real> <constrained/none>, it will be able to solve single-objective continuous optimization problems with or without constraints. On the other hand, the label sets <single> <real> mean that the algorithm can only solve unconstrained problems, the label sets <single> <real> <constrained> mean that the algorithm can only solve constrained problems, and the label sets <single> <real/binary> mean that the algorithm can solve problems with either real variables or binary variables.

Each algorithm, benchmark problem, and metric should be tagged with at least one

label, otherwise it will not be appeared in the lists in the GUI. After selecting multiple labels in the GUI, only the algorithms, benchmark problems, and metrics containing the same labels will be appeared. Details of the label based filter strategy can be found *here*. The labels of all the algorithms and benchmark problems in PlatEMO are referred to *List of Algorithms* and *List of Problems*, respectively.

IV. Extending PlatEMO

A. *ALGORITHM Class*

An algorithm should be written as a subclass of `ALGORITHM` and put in the folder `PlatEMO\Algorithms`, which contains the following properties and methods:

Property	Specified by	Description
`parameter`	Users	Parameters of the algorithm
`save`	Users	Number of populations saved in an execution
`outputFcn`	Users	Function called in `NotTerminated()`
`pro`	`Solve()`	Problem solved in current execution
`result`	`NotTerminated()`	Populations saved in current execution
`metric`	`NotTerminated()`	Metric values of saved populations
Method	**Be redefined**	**Description**
`ALGORITHM`	Cannot	Set the properties specified by users Input: Parameter settings like `'Name',Value,…` Output: `ALGORITHM` object
`Solve`	Cannot	Solve a problem via the algorithm Input: `PROBLEM` object Output: None
`main`	Must	Main procedure of the algorithm Input: `PROBLEM` object Output: None
`NotTerminated`	Cannot	Function called before each iteration in `main()` Input: An array of `SOLUTION` objects, i.e., a population Output: Whether the algorithm terminates (logical)
`ParameterSet`	Cannot	Set the parameter values according to `parameter` Input: Default parameter settings Output: User-specified parameter settings

Each algorithm should inherit `ALGORITHM` and redefine the method `main()`. For example, the code of `GA.m` is

```
1 classdef GA < ALGORITHM
2 % <single><real/integer/label/binary/permutation><large/none><constrained/none>
3 % Genetic algorithm
4 % proC ---  1 --- Probability of crossover
5 % disC --- 20 --- Distribution index of crossover
6 % proM ---  1 --- Expectation of the number of mutated variables
```

```
7  % disM --- 20 --- Distribution index of mutation
8
9  %------------------------ Reference -----------------------
10 % J. H. Holland, Adaptation in Natural and Artificial
11 % Systems, MIT Press, 1992.
12 %----------------------------------------------------------
13
14     methods
15         function main(Alg,Pro)
16             [proC,disC,proM,disM] = Alg.ParameterSet(1,20,1,20);
17             P = Pro.Initialization();
18             while Alg.NotTerminated(P)
19                 Q = TournamentSelection(2,Pro.N,FitnessSingle(P));
20                 O = OperatorGA(P(Q),{proC,disC,proM,disM});
21                 P = [P,O];
22                 [~,rank] = sort(FitnessSingle(P));
23                 P = P(rank(1:Pro.N));
24             end
25         end
26     end
27 end
```

The functions of each line are as follows:

Line 1: Inheriting the `ALGORITHM` class;

Line 2: Tagging the algorithm with labels (see *Labels of Algorithms, Problems, and Metrics* for details);

Line 3: Full name of the algorithm;

Lines 4-7: Parameter name --- default value --- description, which are shown in the parameter setting list in the GUI;

Lines 9-12: Reference of the algorithm;

Line 15: Redefining the method of main procedure;

Line 16: Obtaining the parameter values specified by users, where `1,20,1,20` are default values of the four parameters `proC,disC,proM,disM`;

Line 17: Obtaining an initial population by calling a method of the problem;

Line 18: Storing the population and checking whether the algorithm terminates; if so, the algorithm will immediately terminate by throwing an error;

Line 19: Binary tournament based mating selection achieved by a public function;

Line 20: Offspring generation achieved by a public function;

Line 21: Combing the current population with the offspring population;

Line 22: Sorting the solutions based on their fitness calculated by a public function;

Line 23: Retaining half the solutions with better fitness for the next iteration.

In the above codes, the functions `ParameterSet()` and `NotTerminated()` are provided by the `ALGORITHM` class, and the function `Initialization()` is provided by the `PROBLEM` class. Besides, the functions `TournamentSelection()`, `FitnessSingle()`, and `OperatorGA()` are public functions in the folder `PlatEMO\Algorithms\Utility functions`. The following table lists the functions that can be used in algorithms, where the details of them are referred to the comments in their codes. Besides, their techniques for efficiency improvement can be found *here*.

Function Name	Description
`ALGORITHM.NotTerminated`	Function called before each iteration of the algorithm, which stores the current population and check whether the algorithm terminates
`ALGORITHM.ParameterSet`	Set the parameter values specified by users
`PROBLEM.Initialization`	Initialize a population for the problem
`PROBLEM.Evaluation`	Evaluate a population and generate an array of `SOLUTION` object
`CrowdingDistance`	Crowding distance calculation for multi-objective optimization
`FitnessSingle`	Fitness calculation for single-objective optimization
`NDSort`	Non-dominated sorting for multi-objective optimization
`OperatorDE`	The variation operator of differential evolution
`OperatorFEP`	The variation operator of fast evolutionary programming
`OperatorGA`	The variation operators of genetic algorithm
`OperatorGAhalf`	The variation operators of genetic algorithm, where only the first half of offspring solutions are returned
`OperatorPSO`	The variation operator of particle swarm optimization
`RouletteWheelSelection`	Roulette-wheel selection
`TournamentSelection`	Tournament selection
`UniformPoint`	Generate a set of uniformly distributed points

B. PROBLEM Class

A problem should be written as a subclass of `PROBLEM` and put in the folder `PlatEMO\Problems`, which contains the following properties and methods:

Property	Specified by	Description
`N`	Users	Population size of algorithms
`M`	Users and `Setting()`	Number of objectives of the problem
`D`	Users and `Setting()`	Number of decision variables of the problem

maxFE	Users	Maximum number of function evaluations
FE	Evaluation()	Number of function evaluations consumed in current execution
maxRuntime	Users	Maximum runtime
encoding	Setting()	Encoding scheme of each variable
lower	Setting()	Lower bound of each variable
upper	Setting()	Upper bound of each variable
optimum	GetOptimum()	Optimal values of the problem, such as the minimum objective value of single-objective optimization problems and a set of points on the Pareto front of multi-objective optimization problems
PF	GetPF()	Pareto front of the problem, such as a 1-D curve of bi-objective optimization problems, a 2-D surface of tri-objective optimization problems, and feasible regions of constrained optimization problems
parameter	Users	Parameters of the problem
Method	**Be redefined**	**Description**
PROBLEM	Cannot	Set the properties specified by users Input: Parameter settings like 'Name',Value,… Output: ALGORITHM object
Setting	Must	Default settings of the problem Input: None Output: None
Initialization	Can	Initialize a population Input: Population size Output: An array of SOLUTION objects, i.e., a population
Evaluation	Can	Evaluate a population and generate solution objects Input: A matrix consisting of decision vectors Output: An array of SOLUTION objects, i.e., a population
CalDec	Can	Repair invalid solutions in a population Input: A matrix consisting of decision vectors Output: A matrix consisting of repaired decision vectors
CalObj	Must	Calculate the objective values of solutions in a population. All objectives are to be minimized Input: A matrix consisting of decision vectors Output: A matrix consisting of objective values
CalCon	Can	Calculate the constraint violations of solutions in a population. A constraint is satisfied if and only if the constraint violation is not positive Input: A matrix consisting of decision vectors Output: A matrix consisting of constraint violations
CalObjGrad	Can	Calculate the gradients of a solution on objectives

		Input: A decision vector Output: A Jacobian matrix
`CalConGrad`	Can	Calculate the gradients of a solution on constraints Input: A decision vector Output: A Jacobian matrix
`GetOptimum`	Can	Generate the optimal values and store in `optimum` Input: The number of optimal values Output: Optimal values (a matrix)
`GetPF`	Can	Generate the Pareto front and store in `PF` Input: None Output: Data for plotting the Pareto front (a matrix or cell)
`CalMetric`	Can	Calculate the metric value of a population Input 1: Metric name Input 2: An array of `SOLUTION` objects, i.e., a population Output: Metric value (scalar)
`DrawDec`	Can	Display the decision variables of a population Input: An array of `SOLUTION` objects, i.e., a population Output: None
`DrawObj`	Can	Display the objective values of a population Input: An array of `SOLUTION` objects, i.e., a population Output: None
`ParameterSet`	Cannot	Set the parameter values according to `parameter` Input: Default parameter settings Output: User-specified parameter settings

Each benchmark problem should inherit `PROBLEM` and redefine the methods `Setting()` and `CalObj()`. For example, the code of `SOP_F1.m` is

```
classdef SOP_F1 < PROBLEM
% <single><real><expensive/none>
% Sphere function

%------------------------ Reference ------------------------
% X. Yao, Y. Liu, and G. Lin, Evolutionary programming made
% faster, IEEE Transactions on Evolutionary Computation,
% 1999, 3(2): 82-102.
%-------------------------------------------------------------

    methods
        function Setting(obj)
            obj.M = 1;
            if isempty(obj.D); obj.D = 30; end
            obj.lower = zeros(1,obj.D) - 100;
```

```
16            obj.upper = zeros(1,obj.D) + 100;
17            obj.encoding = ones(1,obj.D);
18        end
19        function PopObj = CalObj(obj,PopDec)
20            PopObj = sum(PopDec.^2,2);
21        end
22    end
23 end
```

The functions of each line are as follows:

Line 1: Inheriting the `PROBLEM` class;

Line 2: Tagging the problem with labels (see *Labels of Algorithms, Problems, and Metrics* for details);

Line 3: Full name of the problem;

Lines 5-9: Reference of the problem;

Line 12: Redefining the method of default parameter settings;

Line 13: Setting the number of objectives;

Line 14: Setting the number of decision variables if it is not specified by users;

Lines 15-16: Setting the lower bounds and upper bounds of decision variables;

Line 17: Setting the encoding schemes of decision variables;

Line 19: Redefining the method of calculating objective values;

Line 20: Calculating the objective values of solutions in a population.

The default method `Initialization()` randomly initializes a population. This method can be redefined to specify a novel initialization strategy. For example, `Sparse_NN.m` initializes a population in which half the decision variables are zero:

```
function Population = Initialization(obj,N)
    if nargin < 2; N = obj.N; end
    PopDec = (rand(N,obj.D)-0.5)*2.*randi([0 1],N,obj.D);
    Population = SOLUTION(PopDec);
end
```

The default method `CalDec()` repairs invalid solutions in a population, where each decision variable will be set to the boundary values if it is larger than the upper bound or smaller than the lower bound. This method can be redefined to specify a novel repair strategy. For example, `MOKP.m` repairs solutions that exceed the capacity, so that no constraint needs to be defined in this problem:

```
function PopDec = CalDec(obj,PopDec)
    C = sum(obj.W,2)/2;
    [~,rank] = sort(max(obj.P./obj.W));
    for i = 1 : size(PopDec,1)
```

```
        while any(obj.W*PopDec(i,:)'>C)
            k = find(PopDec(i,rank),1);
            PopDec(i,rank(k)) = 0;
        end
    end
end
```

The default method `CalCon()` returns zero as the constraint violation of the solutions in a population, i.e., all the solutions are feasible. This method can be redefined to specify constraint functions for the problem. For example, `CF4.m` calculates a constraint for each solution:

```
function PopCon = CalCon(obj,X)
    t = X(:,2)-sin(6*pi*X(:,1)+2*pi/size(X,2))-0.5*X(:,1)+0.25;
    PopCon = -t./(1+exp(4*abs(t)));
end
```

Use `all(PopCon<=0,2)` to determine whether each solution is feasible or not. Note that equality constraints should be converted into inequality constraints, the details of which can be found in Section 3.2 of *this paper*. The default method `Evaluation()` calls `CalDec()`, `CalObj()`, and `CalCon()` in sequence to instantiate `SOLUTION` objects, and also adds the number of consumed function evaluations `FE`. This method can be redefined to perform solution repair, objective calculation, and constraint calculation in a single function, where `CalDec()`, `CalObj()`, and `CalCon()` will not be called anymore. For example, `MW2.m` calculates objective values and constraint violations in a single function:

```
function Population = Evaluation(obj,varargin)
    X = varargin{1};
    X=max(min(X,repmat(obj.upper,size(X,1),1)),repmat(obj.lower,size(X,1),1));
    z=1-exp(-10*(X(:,obj.M:end)-(repmat(obj.M:obj.D,size(X,1),1)-1)/obj.D).^2);
    g = 1+sum((1.5+(0.1/obj.D)*z.^2-1.5*cos(2*pi*z)),2);
    PopObj(:,1) = X(:,1);
    PopObj(:,2) = g.*(1-PopObj(:,1)./g);
    L = sqrt(2)*PopObj(:,2)-sqrt(2)*PopObj(:,1);
    PopCon = sum(PopObj,2)-1-0.5*sin(3*pi*l).^8;
    Population = SOLUTION(X,PopObj,PopCon,varargin{2:end});
    obj.FE = obj.FE+length(Population);
end
```

The default method `CalObjGrad()` estimates the gradients of objectives via finite difference, while this method can be redefined to calculate gradients more accurately. Similarly, the default method `CalConGrad()` estimates the gradients of constraints via finite difference, while this method can be redefined to calculate gradients more

accurately. The method `GetOptimum()` can be redefined to specify the optimal values of the problem, which are used for metric calculation. For example, `SOP_F8.m` returns the optimal value of the objective function:

```
function R = GetOptimum(obj,N)
    R = -418.9829*obj.D;
end
```

and `DTLZ2.m` returns a set of uniformly distributed points on the Pareto front:

```
function R = GetOptimum(obj,N)
    R = UniformPoint(N,obj.M);
    R = R./repmat(sqrt(sum(R.^2,2)),1,obj.M);
end
```

The strategies for sampling points on different Pareto fronts can be found *here*. The method `GetPF()` can be redefined to specify the Pareto front or feasible regions of multi-objective optimization problems for the visualization achieved in `DrawObj()`. For example, `DTLZ2.m` returns the data for plotting the 2-D and 3-D Pareto fronts:

```
function R = GetPF(obj)
    if obj.M == 2
        R = obj.GetOptimum(100);
    elseif obj.M == 3
        a = linspace(0,pi/2,10)';
        R = {sin(a)*cos(a'),sin(a)*sin(a'),cos(a)*ones(size(a'))};
    else
        R = [];
    end
end
```

and `MW1.m` returns the data for plotting the feasible regions:

```
function R = GetPF(obj)
    [x,y]= meshgrid(linspace(0,1,400),linspace(0,1.5,400));
    z    = nan(size(x));
    fes  = x+y-1-0.5*sin(2*pi*(sqrt(2)*y-sqrt(2)*x)).^8 <= 0;
    z(fes&0.85*x+y>=1) = 0;
    R    = {x,y,z};
end
```

The default method `CalMetric()` feeds a population and the optimal values `optimum` to a metric function to calculate the metric value. This method can be redefined to feed different variables to metric functions. For example, `SMMOP1.m` feeds the Pareto optimal set rather than the points on the Pareto front when calculating the metric value

of IGDX:

```
function score = CalMetric(obj,metName,Population)
    switch metName
        case 'IGDX'
            score = feval(metName,Population,obj.POS);
        otherwise
            score = feval(metName,Population,obj.optimum);
    end
end
```

The method `DrawDec()` displays the decision variables of a population, which is used for the visualization of results in the GUI. This method can be redefined to specify a novel visualization method. For example, `TSP.m` displays the route of the best solution:

```
function DrawDec(obj,P)
    [~,best] = min(P.objs);
    Draw(obj.R(P(best).dec([1:end,1]),:),'-k','LineWidth',1.5);
    Draw(obj.R);
end
```

The method `DrawObj()` displays the objective values of a population, which is used for the visualization of results in the GUI. This method can be redefined to specify a novel visualization method. For example, `Sparse_CD.m` adds labels to the axes:

```
function DrawObj(obj,P)
    Draw(P.objs,{'Kernel k-means','Ratio cut',[]});
end
```

where `Draw()` is a function in the folder `PlatEMO\GUI` for displaying data.

C. SOLUTION Class

A `SOLUTION` object denotes an individual, and an array of `SOLUTION` objects denote a population. The `SOLUTION` class contains the following properties and methods:

Property	Specified by	Description
`dec`	Users	Decision variables of the solution
`obj`	`SOLUTION()`	Objective values of the solution
`con`	`SOLUTION()`	Constraint violations of the solution
`add`	`adds()`	Additional properties (e.g., velocity) of the solution
Method	**Description**	
`SOLUTION`	Generate `SOLUTION` objects Input 1: A matrix consisting of decision vectors Input 2: A matrix consisting of objective values	

	Input 3: A matrix consisting of constraint violations Input 4: A matrix consisting of additional properties Output: An array of `SOLUTION` objects
`decs`	Get the decision variables of multiple solutions Input: None Output: A matrix consisting of decision vectors
`objs`	Get the objective values of multiple solutions Input: None Output: A matrix consisting of objective values
`cons`	Get the constraint violations of multiple solutions Input: None Output: A matrix consisting of constraint violations
`adds`	Set and get the additional properties of multiple solutions Input: Default additional properties Output: A matrix consisting of additional properties
`best`	Get the feasible and best solution for single-objective optimization, or the feasible and non-dominated solutions for multi-objective optimization Input: None Output: A subarray of best `SOLUTION` objects in the population

For example, the following code generates a population with ten solutions, then gets the objective matrix of the best solutions in the population:

```
Population = SOLUTION(rand(10,5),rand(10,1),zeros(10,1));
BestObjs   = Population.best.objs
```

Note that `SOLUTION()` should be called only in the method `Evaluation()` of `PROBLEM` class.

D. Whole Procedure of One Run

The following code uses the genetic algorithm to solve the sphere function:

```
Alg = GA();
Pro = SOP_F1();
Alg.Solve(Pro);
```

where the functions called in the execution of `Alg.Solve(Pro)` are as follows.

E. Metric Function

A metric should be written as a function and put in the folder `PlatEMO\Metrics`. For example, the code of `IGD.m` is

```
function score = IGD(Population,optimum)
% <min> <multi/many> <real/integer/label/binary/permutation>
<large/none>       <constrained/none>       <expensive/none>
<multimodal/none> <sparse/none> <dynamic/none> <robust/none>
% Inverted generational distance

%------------------------ Reference -----------------------
% C. A. Coello Coello and N. C. Cortes, Solving
% multiobjective optimization problem using an artificial
% immune system, Genetic Programming and Evolvable
```

```
9  % Machines, 2005, 6(2): 163-190.
10 %--------------------------------------------------------------
11
12     PopObj = Population.best.objs;
13     if size(PopObj,2) ~= size(optimum,2)
14         score = nan;
15     else
16         score = mean(min(pdist2(optimum,PopObj),[],2));
17     end
18 end
```

The functions of each line are as follows:

Line 1: Function declaration, where the first input is a population (i.e., an array of `SOLUTION` objects), the second input is the optimums of a problem (i.e., the `optimum` property of the problem), and the output is the metric value;

Line 2: Tagging the metric with labels (see *Labels of Algorithms, Problems, and Metrics* for details); note that `<min>` or `<max>` should be the first label;

Line 3: Full name of the metric;

Lines 5-10: Reference of the metric;

Line 12: Obtaining the feasible and non-dominated solutions in the population;

Lines 13-14: Returns `nan` if there is no feasible solution in the population;

Lines 15-16: Returns the IGD value of the feasible and non-dominated solutions.

V. List of Algorithms

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
1	ABC	Artificial bee colony algorithm	√			√	√				√	√							
2	AB-SAEA	Adaptive Bayesian based surrogate-assisted evolutionary algorithm		√	√	√	√						√						
3	ACO	Ant colony optimization	√							√	√								
4	Adam	Adaptive moment estimation	√			√					√								
5	AdaW	Evolutionary algorithm with adaptive weights		√	√	√	√	√	√	√									
6	AGE-II	Approximation-guided evolutionary multi-objective algorithm II		√		√	√	√	√	√									
7	AGE-MOEA	Adaptive geometry estimation-based many-objective evolutionary algorithm		√	√	√	√	√	√	√		√							
8	AGE-MOEA-II	Adaptive geometry estimation-based many-objective evolutionary algorithm II		√	√	√	√	√	√	√		√							
9	A-NSGA-III	Adaptive NSGA-III		√	√	√	√	√	√	√		√							
10	AR-MOEA	Adaptive reference points based multi-objective evolutionary algorithm		√	√	√	√	√	√	√		√							
11	BCE-IBEA	Bi-criterion evolution based IBEA		√	√	√	√	√	√	√									
12	BCE-MOEA/D	Bi-criterion evolution based MOEA/D		√	√	√	√	√	√	√									
13	BFGS	A quasi-Newton method proposed by Broyden, Fletcher, Goldfarb, and Shanno	√			√					√								
14	BiCo	Bidirectional coevolution constrained multiobjective evolutionary algorithm		√		√	√	√	√	√		√							
15	BiGE	Bi-goal evolution			√	√	√	√	√	√									
16	BLEAQII	Bilevel evolutionary algorithm based on quadratic approximations II		√		√						√						√	
17	BSPGA	Binary space partition tree based genetic algorithm	√						√		√	√							
18	C3M	Constraint, multiobjective, multi-stage, multi-constraint evolutionary algorithm		√		√	√	√	√	√		√							
19	CAEAD	Dual-population evolutionary algorithm based on alternative evolution and degeneration		√		√	√	√	√	√		√							
20	CA-MOEA	Clustering based adaptive multi-objective evolutionary algorithm		√		√	√	√	√	√									
21	CCGDE3	Cooperative coevolution GDE3		√		√	√				√								
22	CCMO	Coevolutionary constrained multi-objective optimization framework		√		√	√	√	√	√		√							
23	c-DPEA	Constrained dual-population evolutionary algorithm		√		√	√	√	√	√		√							
24	CLIA	Evolutionary algorithm with cascade clustering and reference point incremental learning		√	√	√	√	√	√	√									
25	CMA-ES	Covariance matrix adaptation evolution strategy	√			√	√				√	√							
26	CMME	Constrained many-objective evolutionary algorithm with enhanced mating and environmental selections		√		√	√	√	√	√		√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
27	CMMO	Coevolutionary multi-modal multi-objective optimization framework		√		√	√	√	√	√				√					
28	CMOCSO	Competitive and cooperative swarm optimization constrained multi-objective optimization algorithm		√		√					√	√							
29	C-MOEA/D	Constraint-MOEA/D		√	√	√	√	√	√	√		√							
30	CMOEA-MS	Constrained multiobjective evolutionary algorithm with multiple stages		√		√	√	√	√	√		√							
31	CMOEMT	Constrained multi-objective optimization based on evolutionary multitasking optimization		√		√						√							
32	CMOPSO	Competitive mechanism based multi-objective particle swarm optimizer		√		√	√												
33	CMOQLMT	Constrained multi-objective optimization based on Q-learning and multitasking		√		√						√							
34	CMOSMA	Constrained multi-objective evolutionary algorithm with self-organizing map		√	√	√	√					√							
35	CNSDE/DVC	Constrained nondominated sorting differential evolution based on decision variable classification		√		√	√												√
36	CPS-MOEA	Classification and Pareto domination based multi-objective evolutionary		√		√	√						√						
37	CSEA	Classification based surrogate-assisted evolutionary algorithm		√	√	√							√						
38	CSO	Competitive swarm optimizer	√			√	√				√	√							
39	C-TAEA	Two-archive evolutionary algorithm for constrained MOPs		√	√	√	√	√	√	√		√							
40	C-TSEA	Constrained two-stage evolutionary algorithm		√	√	√	√	√	√	√		√							
41	DAEA	Duplication analysis based evolutionary algorithm		√					√										
42	DCNSGA-III	Dynamic constrained NSGA-III		√	√	√	√	√	√	√		√							
43	DE	Differential evolution	√			√	√				√	√							
44	DEA-GNG	Decomposition based evolutionary algorithm guided by growing neural gas		√	√	√	√	√	√	√									
45	DGEA	Direction guided evolutionary algorithm		√	√	√	√				√								
46	DMOEA-eC	Decomposition-based multi-objective evolutionary algorithm with the e-constraint framework		√		√	√	√	√	√									
47	dMOPSO	MOPSO based on decomposition		√		√	√												
48	DN-NSGA-II	Decision space based niching NSGA-II		√		√	√							√					
49	DNSGA-II	Dynamic NSGA-II		√		√	√	√	√	√						√			
50	DP-PPS	Tri-population based push and pull search		√		√						√							
51	DSPCMDE	Dynamic selection preference-assisted constrained multiobjective differential evolution		√		√	√					√							
52	DWU	Dominance-weighted uniformity multi-objective evolutionary algorithm		√		√	√	√	√	√									
53	EAG-MOEA/D	External archive guided MOEA/D		√		√	√	√	√	√									
54	EDN-ARMOEA	Efficient dropout neural network based AR-MOEA		√	√	√	√						√						
55	EFR-RR	Ensemble fitness ranking with a ranking restriction scheme		√	√	√	√	√	√	√									
56	EGO	Efficient global optimization	√			√	√						√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
57	EIM-EGO	Expected improvement matrix based efficient global optimization		√		√	√						√						
58	EMCMO	Evolutionary multitasking-based constrained multiobjective optimization		√		√	√	√	√	√		√							
59	e-MOEA	Epsilon multi-objective evolutionary algorithm		√	√	√	√	√	√	√									
60	EMyO/C	Evolutionary many-objective optimization algorithm with clustering-based		√	√	√	√												
61	ENS-MOEA/D	Ensemble of different neighborhood sizes based MOEA/D		√	√	√	√												
62	FDV	Fuzzy decision variable framework with various internal optimizers		√	√	√	√				√								
63	FEP	Fast evolutionary programming	√			√	√				√	√							
64	FLEA	Fast sampling based evolutionary algorithm		√	√	√					√								
65	FRCG	Fletcher-Reeves conjugate gradient	√			√					√								
66	FRCGM	Fletcher-Reeves conjugate gradient (for multi-objective optimization)		√	√	√					√	√							
67	FROFI	Feasibility rule with the incorporation of objective function information	√			√	√				√	√							
68	GA	Genetic algorithm	√			√	√	√	√	√	√	√							
69	GDE3	Generalized differential evolution 3		√		√	√					√							
70	GFM-MOEA	Generic front modeling based multi-objective evolutionary algorithm		√	√	√	√	√	√	√									
71	GLMO	Grouped and linked mutation operator algorithm		√		√	√				√								
72	g-NSGA-II	g-dominance based NSGA-II		√		√	√	√	√	√									
73	GPSO	Gradient based particle swarm optimization algorithm	√			√					√	√							
74	GPSOM	Gradient based particle swarm optimization algorithm (for multi-objective optimization)		√	√	√					√	√							
75	GrEA	Grid-based evolutionary algorithm			√	√	√	√	√	√									
76	HeE-MOEA	Multiobjective evolutionary algorithm with heterogeneous ensemble based infill criterion		√		√	√						√						
77	HHC-MMEA	Hybrid hierarchical clustering based multi-modal multi-objective evolutionary algorithm		√		√					√			√	√				
78	hpaEA	Hyperplane assisted evolutionary algorithm		√	√	√	√	√	√	√									
79	HREA	Hierarchy ranking based evolutionary algorithm		√		√	√							√					
80	HypE	Hypervolume estimation algorithm		√	√	√	√	√	√	√									
81	IBEA	Indicator-based evolutionary algorithm		√	√	√	√	√	√	√									
82	ICMA	Indicator based constrained multi-objective algorithm		√		√	√					√							
83	I-DBEA	Improved decomposition-based evolutionary algorithm		√	√	√	√	√	√	√		√							
84	IM-MOEA	Inverse modeling based multiobjective evolutionary algorithm		√		√	√				√								
85	IM-MOEA/D	Inverse modeling multiobjective evolutionary algorithm based on decomposition		√		√	√				√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
86	IMODE	Improved multi-operator differential evolution	√			√	√				√	√							
87	I-SIBEA	Interactive simple indicator-based evolutionary algorithm		√		√	√	√	√	√									
88	Izui	An aggregative gradient based multi-objective optimizer proposed by Izui et al.		√	√	√					√	√							
89	KnEA	Knee point driven evolutionary algorithm			√	√	√	√	√	√		√							
90	K-RVEA	Surrogate-assisted RVEA		√	√	√	√						√						
91	KTA2	Kriging-assisted Two_Arch2		√	√	√	√						√						
92	LCSA	Linear combination-based search algorithm		√	√	√	√				√								
93	LERD	Large-scale evolutionary algorithm with reformulated decision variable analysis		√	√	√					√								
94	LMEA	Evolutionary algorithm for large-scale many-objective optimization		√	√	√	√				√								
95	LMOCSO	Large-scale multi-objective competitive swarm optimization algorithm		√	√	√	√				√	√							
96	LMOEA-DS	Large-scale evolutionary multi-objective optimization assisted by directed sampling		√		√	√				√								
97	LMPFE	Evolutionary algorithm with local model based Pareto front estimation		√	√	√	√	√	√	√									
98	LSMOF	Large-scale multi-objective optimization framework with NSGA-II		√		√	√				√								
99	MaOEA-CSS	Many-objective evolutionary algorithms based on coordinated selection		√	√	√	√	√	√	√									
100	MaOEA-DDFC	Many-objective evolutionary algorithm based on directional diversity and favorable convergence		√	√	√	√	√	√	√									
101	MaOEA/IGD	IGD based many-objective evolutionary algorithm			√	√	√	√	√	√									
102	MaOEA/IT	Many-objective evolutionary algorithms based on an independent two-stage		√	√	√	√					√							
103	MaOEA-R&D	Many-objective evolutionary algorithm based on objective space reduction			√	√	√	√	√	√									
104	MCEA/D	Multiple classifiers-assisted evolutionary algorithm based on decomposition		√	√	√	√						√						
105	MFEA	Multifactorial evolutionary algorithm	√			√	√	√	√	√	√						√		
106	MFEA-II	Multifactorial evolutionary algorithm II	√			√	√	√	√	√	√						√		
107	MMEA-WI	Weighted indicator-based evolutionary algorithm for multimodal multi-objective optimization		√		√	√							√					
108	MMOPSO	MOPSO with multiple search strategies		√		√	√												
109	MO_Ring_PSO_SCD	Multiobjective PSO using ring topology and special crowding distance		√		√	√							√					
110	MOCell	Cellular genetic algorithm		√		√	√	√	√	√		√							
111	MOCGDE	Multi-objective conjugate gradient and differential evolution algorithm		√	√	√					√	√							
112	MO-CMA	Multi-objective covariance matrix adaptation evolution strategy		√		√	√												
113	MOEA/D	Multiobjective evolutionary algorithm based on decomposition		√	√	√	√	√	√	√									

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
114	MOEA/D-AWA	MOEA/D with adaptive weight adjustment		√	√	√	√	√	√	√									
115	MOEA/D-CMA	MOEA/D with covariance matrix adaptation evolution strategy		√	√	√	√												
116	MOEA/DD	Many-objective evolutionary algorithm based on dominance and decomposition		√	√	√	√	√	√	√		√							
117	MOEA/D-DAE	MOEA/D with detect-and-escape strategy		√		√	√	√	√	√		√							
118	MOEA/D-DCWV	MOEA/D with distribution control of weight vector set		√	√	√	√	√	√	√									
119	MOEA/D-DE	MOEA/D based on differential evolution		√	√	√	√												
120	MOEA/D-DRA	MOEA/D with dynamical resource allocation		√	√	√	√												
121	MOEA/D-DU	MOEA/D with a distance based updating strategy		√	√	√	√	√	√	√									
122	MOEA/D-DYTS	MOEA/D with dynamic Thompson sampling		√	√	√	√												
123	MOEA/D-EGO	MOEA/D with efficient global optimization		√		√	√						√						
124	MOEA/D-FRRMAB	MOEA/D with fitness-rate-rank-based multiarmed bandit		√	√	√	√												
125	MOEA/D-M2M	MOEA/D based on MOP to MOP		√		√	√												
126	MOEA/D-MRDL	MOEA/D with maximum relative diversity loss		√		√	√												
127	MOEA/D-PaS	MOEA/D with Pareto adaptive scalarizing approximation		√	√	√	√												
128	MOEA/D-PFE	MOEA/D with Pareto front estimation		√	√	√	√	√	√	√									
129	MOEA/D-STM	MOEA/D with stable matching		√	√	√	√												
130	MOEA/D-UR	MOEA/D with update when required		√	√	√	√	√	√	√									
131	MOEA/D-URAW	MOEA/D with uniform randomly adaptive weights		√	√	√	√	√	√	√									
132	MOEA/DVA	Multi-objective evolutionary algorithm based on decision variable		√		√	√				√								
133	MOEA/D-VOV	MOEA/D with virtual objective vectors		√	√	√	√	√	√	√									
134	MOEA/IGD-NS	Multi-objective evolutionary algorithm based on an enhanced IGD		√		√	√	√	√	√									
135	MOEA-PC	Multiobjective evolutionary algorithm based on polar coordinates		√		√	√												
136	MOEA/PSL	Multi-objective evolutionary algorithm based on Pareto optimal subspace		√		√	√		√		√	√			√				
137	MOEA-RE	Multi-objective evolutionary algorithm with robustness enhancement		√		√	√	√	√	√									√
138	MO-EGS	Multi-objective evolutionary gradient search		√		√					√								
139	MOMBI-II	Many objective metaheuristic based on the R2 indicator II		√	√	√	√	√	√	√									
140	MO-MFEA	Multi-objective multifactorial evolutionary algorithm		√		√	√	√	√	√		√					√		
141	MO-MFEA-II	Multi-objective multifactorial evolutionary algorithm II		√		√	√	√	√	√		√					√		
142	MOPSO	Multi-objective particle swarm optimization		√		√	√												

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
143	MOPSO-CD	MOPSO with crowding distance		√		√	√												
144	MOSD	Multiobjective steepest descent		√		√					√	√							
145	M-PAES	Memetic algorithm with Pareto archived evolution strategy		√		√	√												
146	MP-MMEA	Multi-population multi-modal multi-objective evolutionary algorithm		√		√	√				√			√	√				
147	MPSO/D	Multi-objective particle swarm optimization algorithm based on decomposition		√	√	√	√												
148	MSCMO	Multi-stage constrained multi-objective evolutionary algorithm		√		√	√	√	√	√		√							
149	MSEA	Multi-stage multi-objective evolutionary algorithm		√		√	√	√	√	√									
150	MSKEA	Multi-stage knowledge-guided evolutionary algorithm		√		√	√		√		√	√			√				
151	MSOPS-II	Multiple single objective Pareto sampling II		√	√	√	√					√							
152	MTCMO	Multitasking constrained multi-objective optimization		√		√	√	√	√	√		√							
153	MTS	Multiple trajectory search		√		√	√												
154	MultiObjective EGO	Multi-objective efficient global optimization		√		√	√					√	√						
155	MyO-DEMR	Many-objective differential evolution with mutation restriction		√	√	√	√												
156	NBLEA	Nested bilevel evolutionary algorithm		√		√						√						√	
157	NelderMead	The Nelder-Mead algorithm	√			√													
158	NMPSO	Novel multi-objective particle swarm optimization		√	√	√	√												
159	NNIA	Nondominated neighbor immune algorithm		√		√	√	√	√	√									
160	NSGA-II	Nondominated sorting genetic algorithm II		√		√	√	√	√	√		√							
161	NSGA-II+ARSBX	NSGA-II with adaptive rotation based simulated binary crossover		√		√	√					√							
162	NSGA-II-conflict	NSGA-II with conflict-based partitioning strategy			√	√	√	√	√	√									
163	NSGA-II-DTI	NSGA-II of Deb's type I robust version		√		√	√	√	√	√		√							√
164	NSGA-III	Nondominated sorting genetic algorithm III		√	√	√	√	√	√	√		√							
165	NSGA-II/SDR	NSGA-II with strengthened dominance relation			√	√	√	√	√	√									
166	NSLS	Multiobjective optimization framework based on nondominated sorting and local search		√		√	√												
167	OFA	Optimal foraging algorithm	√			√	√				√	√							
168	one-by-one EA	Many-objective evolutionary algorithm using a one-by-one selection		√	√	√	√	√	√	√									
169	OSP-NSDE	Non-dominated sorting differential evolution with prediction in the objective space		√		√	√												
170	ParEGO	Efficient global optimization for Pareto optimization		√		√	√						√						
171	PB-NSGA-III	NSGA-III based on Pareto based bi-indicator infill sampling criterion		√	√	√	√						√						
172	PB-RVEA	RVEA based on Pareto based bi-indicator infill sampling criterion		√	√	√	√						√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
173	PC-SAEA	Pairwise comparison based surrogate-assisted evolutionary algorithm		√	√								√						
174	PeEA	Pareto front shape estimation based evolutionary algorithm		√	√	√	√	√	√	√									
175	PESA-II	Pareto envelope-based selection algorithm II		√		√	√	√	√	√									
176	PICEA-g	Preference-inspired coevolutionary algorithm with goals		√	√	√	√	√	√	√									
177	PM-MOEA	Pattern mining based multi-objective evolutionary algorithm		√		√	√		√		√	√			√				
178	POCEA	Paired offspring generation based constrained evolutionary algorithm		√		√	√				√	√							
179	PPS	Push and pull search algorithm		√	√	√	√					√							
180	PREA	Promising-region based EMO algorithm		√	√	√	√	√	√	√									
181	PSO	Particle swarm optimization	√			√	√				√	√							
182	REMO	Expensive multiobjective optimization by relation learning and prediction		√	√	√							√						
183	RM-MEDA	Regularity model-based multiobjective estimation of distribution		√		√	√												
184	RMOEA/DVA	Robust multi-objective evolutionary algorithm with decision variable assortment		√		√	√												√
185	RMSProp	Root mean square propagation	√			√					√								
186	r-NSGA-II	r-dominance based NSGA-II		√		√	√	√	√	√									
187	RPD-NSGA-II	Reference point dominance-based NSGA-II		√	√	√	√	√	√	√									
188	RPEA	Reference points-based evolutionary algorithm			√	√	√	√	√	√									
189	RSEA	Radial space division based evolutionary algorithm		√	√	√	√	√	√	√									
190	RVEA	Reference vector guided evolutionary algorithm		√	√	√	√	√	√	√		√							
191	RVEAa	RVEA embedded with the reference vector regeneration strategy			√	√	√	√	√	√									
192	RVEA-iGNG	RVEA based on improved growing neural gas		√	√	√	√	√	√	√									
193	S3-CMA-ES	Scalable small subpopulations based covariance matrix adaptation		√	√	√	√				√								
194	SA	Simulated annealing	√			√	√				√	√							
195	SACC-EAM-II	Surrogate-assisted cooperative co-evolutionary algorithm of Minamo	√			√	√						√						
196	SACOSO	Surrogate-assisted cooperative swarm optimization	√			√	√				√		√						
197	SADE-Sammon	Sammon mapping assisted differential evolution	√			√	√						√						
198	SAMSO	Multiswarm-assisted expensive optimization	√			√	√				√		√						
199	S-CDAS	Self-controlling dominance area of solutions			√	√	√	√	√	√									
200	SD	Steepest descent	√			√					√								
201	S-ECSO	Enhanced competitive swarm optimizer for sparse optimization		√		√					√				√				
202	SGEA	Steady-state and generational evolutionary algorithm		√		√	√	√	√	√		√				√			

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
203	SGECF	Sparsity-guided elitism co-evolutionary framework		√		√			√		√	√			√				
204	SHADE	Success-history based adaptive differential evolution	√			√	√				√	√							
205	SIBEA	Simple indicator-based evolutionary algorithm		√		√	√	√	√	√									
206	SIBEA-kEMOSS	SIBEA with minimum objective subset of size k with minimum error			√	√	√	√	√	√									
207	SLMEA	Super-large-scale multi-objective evolutionary algorithm		√		√	√		√		√	√			√				
208	SMEA	Self-organizing multiobjective evolutionary algorithm		√		√	√												
209	SMOA	Supervised multi-objective optimization algorithm		√		√							√						
210	SMPSO	Speed-constrained multi-objective particle swarm optimization		√		√	√												
211	SMS-EGO	S metric selection based efficient global optimization		√		√	√						√						
212	SMS-EMOA	S metric selection based evolutionary multiobjective optimization		√		√	√	√	√	√									
213	SparseEA	Evolutionary algorithm for sparse multi-objective optimization problems		√		√	√		√		√	√			√				
214	SparseEA2	Improved SparseEA		√		√	√		√		√	√			√				
215	SPEA2	Strength Pareto evolutionary algorithm 2		√		√	√	√	√	√									
216	SPEA2+SDE	SPEA2 with shift-based density estimation			√	√	√	√	√	√									
217	SPEA/R	Strength Pareto evolutionary algorithm based on reference direction		√	√	√	√	√	√	√									
218	SQP	Sequential quadratic programming	√			√					√	√							
219	SRA	Stochastic ranking algorithm			√	√	√	√	√	√									
220	t-DEA	theta-dominance based evolutionary algorithm		√	√	√	√	√	√	√									
221	TiGE-2	Tri-Goal Evolution Framework for CMaOPs			√	√	√	√	√	√		√							
222	ToP	Two-phase framework with NSGA-II		√		√	√					√							
223	TriMOEA-TA&R	Multi-modal MOEA using two-archive and recombination strategies		√		√	√							√					
224	TriP	Tri-population based coevolutionary algorithm		√	√	√	√					√							
225	TS-NSGA-II	Two stage NSGA-II		√	√	√	√	√	√	√									
226	TSTI	Two-stage evolutionary algorithm with three indicators		√		√	√	√	√	√		√							
227	Two_Arch2	Two-archive algorithm 2		√	√	√	√	√	√	√									
228	URCMO	Utilizing the relationship between constrained and unconstrained Pareto fronts for constrained multi-objective optimization		√		√	√					√							
229	VaEA	Vector angle based evolutionary algorithm		√	√	√	√	√	√	√									
230	WOF	Weighted optimization framework		√		√	√				√								
231	WV-MOEA-P	Weight vector based multi-objective optimization algorithm with preference		√		√	√												

VI. List of Problems

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
1	BT1	Benchmark MOP with bias feature		√		√					√								
2	BT2	Benchmark MOP with bias feature		√		√					√								
3	BT3	Benchmark MOP with bias feature		√		√					√								
4	BT4	Benchmark MOP with bias feature		√		√					√								
5	BT5	Benchmark MOP with bias feature		√		√					√								
6	BT6	Benchmark MOP with bias feature		√		√					√								
7	BT7	Benchmark MOP with bias feature		√		√					√								
8	BT8	Benchmark MOP with bias feature		√		√					√								
9	BT9	Benchmark MOP with bias feature		√		√					√								
10	C10MOP1	Neural architecture search on CIFAR-10		√		√					√		√						
11	C10MOP2	Neural architecture search on CIFAR-10		√		√					√		√						
12	C10MOP3	Neural architecture search on CIFAR-10		√		√					√		√						
13	C10MOP4	Neural architecture search on CIFAR-10		√		√					√		√						
14	C10MOP5	Neural architecture search on CIFAR-10		√		√					√		√						
15	C10MOP6	Neural architecture search on CIFAR-10		√		√					√		√						
16	C10MOP7	Neural architecture search on CIFAR-10		√		√					√		√						
17	C10MOP8	Neural architecture search on CIFAR-10		√		√					√		√						
18	C10MOP9	Neural architecture search on CIFAR-10		√		√					√		√						
19	CEC2008_F1	Shifted sphere function	√			√					√		√						
20	CEC2008_F2	Shifted Schwefel's function	√			√					√		√						
21	CEC2008_F3	Shifted Rosenbrock's function	√			√					√		√						
22	CEC2008_F4	Shifted Rastrign's function	√			√					√		√						
23	CEC2008_F5	Shifted Griewank's function	√			√					√		√						
24	CEC2008_F6	Shifted Ackley's function	√			√					√		√						
25	CEC2008_F7	FastFractal 'DoubleDip' function	√			√					√		√						
26	CEC2010_F1	CEC'2010 constrained optimization benchmark problem	√			√						√							
27	CEC2010_F2	CEC'2010 constrained optimization benchmark problem	√			√						√							
28	CEC2010_F3	CEC'2010 constrained optimization benchmark problem	√			√						√							
29	CEC2010_F4	CEC'2010 constrained optimization benchmark problem	√			√						√							
30	CEC2010_F5	CEC'2010 constrained optimization benchmark problem	√			√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
31	CEC2010_F6	CEC'2010 constrained optimization benchmark problem	√			√						√							
32	CEC2010_F7	CEC'2010 constrained optimization benchmark problem	√			√						√							
33	CEC2010_F8	CEC'2010 constrained optimization benchmark problem	√			√						√							
34	CEC2010_F9	CEC'2010 constrained optimization benchmark problem	√			√						√							
35	CEC2010_F10	CEC'2010 constrained optimization benchmark problem	√			√						√							
36	CEC2010_F11	CEC'2010 constrained optimization benchmark problem	√			√						√							
37	CEC2010_F12	CEC'2010 constrained optimization benchmark problem	√			√						√							
38	CEC2010_F13	CEC'2010 constrained optimization benchmark problem	√			√						√							
39	CEC2010_F14	CEC'2010 constrained optimization benchmark problem	√			√						√							
40	CEC2010_F15	CEC'2010 constrained optimization benchmark problem	√			√						√							
41	CEC2010_F16	CEC'2010 constrained optimization benchmark problem	√			√						√							
42	CEC2010_F17	CEC'2010 constrained optimization benchmark problem	√			√						√							
43	CEC2010_F18	CEC'2010 constrained optimization benchmark problem	√			√						√							
44	CEC2013_F1	Shifted elliptic function	√			√					√								
45	CEC2013_F2	Shifted Rastrigin's function	√			√					√								
46	CEC2013_F3	Shifted Ackley's function	√			√					√								
47	CEC2013_F4	7-nonseparable, 1-separable shifted and rotated elliptic function	√			√					√								
48	CEC2013_F5	7-nonseparable, 1-separable shifted and rotated Rastrigin's function	√			√					√								
49	CEC2013_F6	7-nonseparable, 1-separable shifted and rotated Ackley's function	√			√					√								
50	CEC2013_F7	7-nonseparable, 1-separable shifted and rotated Schwefel's function	√			√					√								
51	CEC2013_F8	20-nonseparable shifted and rotated elliptic function	√			√					√								
52	CEC2013_F9	20-nonseparable shifted and rotated Rastrigin's function	√			√					√								
53	CEC2013_F10	20-nonseparable shifted and rotated Rastrigin's function	√			√					√								
54	CEC2013_F11	20-nonseparable shifted and rotated Schwefel's function	√			√					√								
55	CEC2013_F12	Shifted Rosenbrock's function	√			√					√								
56	CEC2013_F13	Shifted Schwefel's function with conforming overlapping subcomponents	√			√					√								
57	CEC2013_F14	Shifted Schwefel's function with conflicting	√			√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		overlapping subcomponents																	
58	CEC2013_F15	Shifted Schwefel's function	√			√					√								
59	CEC2017_F1	CEC'2017 constrained optimization benchmark problem	√			√						√							
60	CEC2017_F2	CEC'2017 constrained optimization benchmark problem	√			√						√							
61	CEC2017_F3	CEC'2017 constrained optimization benchmark problem	√			√						√							
62	CEC2017_F4	CEC'2017 constrained optimization benchmark problem	√			√						√							
63	CEC2017_F5	CEC'2017 constrained optimization benchmark problem	√			√						√							
64	CEC2017_F6	CEC'2017 constrained optimization benchmark problem	√			√						√							
65	CEC2017_F7	CEC'2017 constrained optimization benchmark problem	√			√						√							
66	CEC2017_F8	CEC'2017 constrained optimization benchmark problem	√			√						√							
67	CEC2017_F9	CEC'2017 constrained optimization benchmark problem	√			√						√							
68	CEC2017_F10	CEC'2017 constrained optimization benchmark problem	√			√						√							
69	CEC2017_F11	CEC'2017 constrained optimization benchmark problem	√			√						√							
70	CEC2017_F12	CEC'2017 constrained optimization benchmark problem	√			√						√							
71	CEC2017_F13	CEC'2017 constrained optimization benchmark problem	√			√						√							
72	CEC2017_F14	CEC'2017 constrained optimization benchmark problem	√			√						√							
73	CEC2017_F15	CEC'2017 constrained optimization benchmark problem	√			√						√							
74	CEC2017_F16	CEC'2017 constrained optimization benchmark problem	√			√						√							
75	CEC2017_F17	CEC'2017 constrained optimization benchmark problem	√			√						√							
76	CEC2017_F18	CEC'2017 constrained optimization benchmark problem	√			√						√							
77	CEC2017_F19	CEC'2017 constrained optimization benchmark problem	√			√						√							
78	CEC2017_F20	CEC'2017 constrained optimization benchmark problem	√			√						√							
79	CEC2017_F21	CEC'2017 constrained optimization benchmark problem	√			√						√							
80	CEC2017_F22	CEC'2017 constrained optimization benchmark problem	√			√						√							
81	CEC2017_F23	CEC'2017 constrained optimization benchmark problem	√			√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
82	CEC2017_F24	CEC'2017 constrained optimization benchmark problem	√			√						√							
83	CEC2017_F25	CEC'2017 constrained optimization benchmark problem	√			√						√							
84	CEC2017_F26	CEC'2017 constrained optimization benchmark problem	√			√						√							
85	CEC2017_F27	CEC'2017 constrained optimization benchmark problem	√			√						√							
86	CEC2017_F28	CEC'2017 constrained optimization benchmark problem	√			√						√							
87	CEC2020_F1	Bent cigar function	√			√													
88	CEC2020_F2	Shifted and rotated Schwefel's function	√			√													
89	CEC2020_F3	Shifted and rotated Lunacek bi-Rastrigin function	√			√													
90	CEC2020_F4	Expanded Rosenbrock's plus Griewangk's function	√			√													
91	CEC2020_F5	Hybrid function 1	√			√													
92	CEC2020_F6	Hybrid function 2	√			√													
93	CEC2020_F7	Hybrid function 3	√			√													
94	CEC2020_F8	Composition function 1	√			√													
95	CEC2020_F9	Composition function 2	√			√													
96	CEC2020_F10	Composition function 3	√			√													
97	CF1	Constrained benchmark MOP		√		√					√	√							
98	CF2	Constrained benchmark MOP		√		√					√	√							
99	CF3	Constrained benchmark MOP		√		√					√	√							
100	CF4	Constrained benchmark MOP		√		√					√	√							
101	CF5	Constrained benchmark MOP		√		√					√	√							
102	CF6	Constrained benchmark MOP		√		√					√	√							
103	CF7	Constrained benchmark MOP		√		√					√	√							
104	CF8	Constrained benchmark MOP		√		√					√	√							
105	CF9	Constrained benchmark MOP		√		√					√	√							
106	CF10	Constrained benchmark MOP		√		√					√	√							
107	CI_HS	Multitasking problem (Griewank function + Rastrigin function)	√			√					√						√		
108	CI_LS	Multitasking problem (Ackley function + Schwefel function)	√			√					√						√		
109	CI_MS	Multitasking problem (Ackley function + Rastrigin function)	√			√					√						√		
110	Community Detection	The community detection problem with label based encoding	√					√			√		√						
111	DAS-CMOP1	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
112	DAS-CMOP2	Difficulty-adjustable and scalable constrained		√		√					√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		benchmark MOP																	
113	DAS-CMOP3	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
114	DAS-CMOP4	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
115	DAS-CMOP5	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
116	DAS-CMOP6	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
117	DAS-CMOP7	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
118	DAS-CMOP8	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
119	DAS-CMOP9	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
120	DOC1	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
121	DOC2	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
122	DOC3	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
123	DOC4	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
124	DOC5	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
125	DOC6	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
126	DOC7	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
127	DOC8	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
128	DOC9	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
129	DTLZ1	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
130	DTLZ2	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
131	DTLZ3	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
132	DTLZ4	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
133	DTLZ5	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
134	DTLZ6	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
135	DTLZ7	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
136	DTLZ8	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√	√	√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
137	DTLZ9	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√	√	√						
138	CDTLZ2	Convex DTLZ2		√	√	√					√		√						
139	IDTLZ1	Inverted DTLZ1		√	√	√					√		√						
140	IDTLZ2	Inverted DTLZ2		√	√	√					√		√						
141	SDTLZ1	Scaled DTLZ1		√	√	√					√		√						
142	SDTLZ2	Scaled DTLZ2		√	√	√					√		√						
143	C1-DTLZ1	Constrained DTLZ1		√	√	√					√	√	√						
144	C1-DTLZ3	Constrained DTLZ3		√	√	√					√	√	√						
145	C2-DTLZ2	Constrained DTLZ2		√	√	√					√	√	√						
146	C3-DTLZ4	Constrained DTLZ4		√	√	√					√	√	√						
147	DC1-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
148	DC1-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
149	DC2-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
150	DC2-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
151	DC3-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
152	DC3-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
153	FCP1	Benchmark constrained MOP proposed by Yuan		√		√						√							
154	FCP2	Benchmark constrained MOP proposed by Yuan		√		√						√							
155	FCP3	Benchmark constrained MOP proposed by Yuan		√		√						√							
156	FCP4	Benchmark constrained MOP proposed by Yuan		√		√						√							
157	FCP5	Benchmark constrained MOP proposed by Yuan		√		√						√							
158	FDA1	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
159	FDA2	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
160	FDA3	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
161	FDA4	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
162	FDA5	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
163	IMMOEA_F1	Benchmark MOP for testing IM-MOEA		√		√					√								
164	IMMOEA_F2	Benchmark MOP for testing IM-MOEA		√		√					√								
165	IMMOEA_F3	Benchmark MOP for testing IM-MOEA		√		√					√								
166	IMMOEA_F4	Benchmark MOP for testing IM-MOEA		√		√					√								
167	IMMOEA_F5	Benchmark MOP for testing IM-MOEA		√		√					√								
168	IMMOEA_F6	Benchmark MOP for testing IM-MOEA		√		√					√								
169	IMMOEA_F7	Benchmark MOP for testing IM-MOEA		√		√					√								
170	IMMOEA_F8	Benchmark MOP for testing IM-MOEA		√		√					√								
171	IMMOEA_F9	Benchmark MOP for testing IM-MOEA		√		√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
172	IMMOEA_F10	Benchmark MOP for testing IM-MOEA		√		√					√								
173	IMOP1	Benchmark MOP with irregular Pareto front		√		√							√						
174	IMOP2	Benchmark MOP with irregular Pareto front		√		√							√						
175	IMOP3	Benchmark MOP with irregular Pareto front		√		√							√						
176	IMOP4	Benchmark MOP with irregular Pareto front		√		√							√						
177	IMOP5	Benchmark MOP with irregular Pareto front		√		√							√						
178	IMOP6	Benchmark MOP with irregular Pareto front		√		√							√						
179	IMOP7	Benchmark MOP with irregular Pareto front		√		√							√						
180	IMOP8	Benchmark MOP with irregular Pareto front		√		√							√						
181	IN1KMOP1	Neural architecture search on ImageNet 1K		√		√					√		√						
182	IN1KMOP2	Neural architecture search on ImageNet 1K		√		√					√		√						
183	IN1KMOP3	Neural architecture search on ImageNet 1K		√		√					√		√						
184	IN1KMOP4	Neural architecture search on ImageNet 1K		√		√					√		√						
185	IN1KMOP5	Neural architecture search on ImageNet 1K		√		√					√		√						
186	IN1KMOP6	Neural architecture search on ImageNet 1K		√		√					√		√						
187	IN1KMOP7	Neural architecture search on ImageNet 1K		√		√					√		√						
188	IN1KMOP8	Neural architecture search on ImageNet 1K		√		√					√		√						
189	IN1KMOP9	Neural architecture search on ImageNet 1K		√		√					√		√						
190	Instance1	Multitasking multi-objective problem (ZDT4-R + ZDT4-G)		√		√					√						√		
191	Instance2	Multitasking multi-objective problem (ZDT4-RC + ZDT4-A)		√		√					√	√					√		
192	KP	The knapsack problem	√						√		√	√							
193	LIR-CMOP1	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
194	LIR-CMOP2	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
195	LIR-CMOP3	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
196	LIR-CMOP4	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
197	LIR-CMOP5	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
198	LIR-CMOP6	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
199	LIR-CMOP7	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
200	LIR-CMOP8	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
201	LIR-CMOP9	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
202	LIR-CMOP10	Constrained benchmark MOP with large infeasible regions		√		√					√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
203	LIR-CMOP11	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
204	LIR-CMOP12	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
205	LIR-CMOP13	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
206	LIR-CMOP14	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
207	LSMOP1	Large-scale benchmark MOP		√	√	√					√								
208	LSMOP2	Large-scale benchmark MOP		√	√	√					√								
209	LSMOP3	Large-scale benchmark MOP		√	√	√					√								
210	LSMOP4	Large-scale benchmark MOP		√	√	√					√								
211	LSMOP5	Large-scale benchmark MOP		√	√	√					√								
212	LSMOP6	Large-scale benchmark MOP		√	√	√					√								
213	LSMOP7	Large-scale benchmark MOP		√	√	√					√								
214	LSMOP8	Large-scale benchmark MOP		√	√	√					√								
215	LSMOP9	Large-scale benchmark MOP		√	√	√					√								
216	MaF1	Inverted DTLZ1		√	√	√					√								
217	MaF2	DTLZ2BZ		√	√	√					√								
218	MaF3	Convex DTLZ3		√	√	√					√								
219	MaF4	Inverted and scaled DTLZ3		√	√	√					√								
220	MaF5	Scaled DTLZ4		√	√	√					√								
221	MaF6	DTLZ5IM		√	√	√					√								
222	MaF7	DTLZ7		√	√	√					√								
223	MaF8	MP-DMP		√	√	√													
224	MaF9	ML-DMP		√	√	√													
225	MaF10	WFG1		√	√	√					√								
226	MaF11	WFG2		√	√	√					√								
227	MaF12	WFG9		√	√	√					√								
228	MaF13	P7		√	√	√					√								
229	MaF14	LSMOP3		√	√	√					√								
230	MaF15	Inverted LSMOP8		√	√	√					√								
231	MaOPP_binary	Many-objective pathfinding problem based on binary encoding			√				√		√		√						
232	MaOPP_real	Many-objective pathfinding problem based on real encoding			√	√					√		√						
233	MLDMP	The multi-line distance minimization problem		√	√	√													
234	MMF1	Multi-modal multi-objective test function		√		√								√					
235	MMF2	Multi-modal multi-objective test function		√		√								√					
236	MMF3	Multi-modal multi-objective test function		√		√								√					

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
237	MMF4	Multi-modal multi-objective test function		√		√								√					
238	MMF5	Multi-modal multi-objective test function		√		√								√					
239	MMF6	Multi-modal multi-objective test function		√		√								√					
240	MMF7	Multi-modal multi-objective test function		√		√								√					
241	MMF8	Multi-modal multi-objective test function		√		√								√					
242	MMMOP1	Multi-modal multi-objective optimization problem		√	√	√								√					
243	MMMOP2	Multi-modal multi-objective optimization problem		√	√	√								√					
244	MMMOP3	Multi-modal multi-objective optimization problem		√	√	√								√					
245	MMMOP4	Multi-modal multi-objective optimization problem		√	√	√								√					
246	MMMOP5	Multi-modal multi-objective optimization problem		√	√	√								√					
247	MMMOP6	Multi-modal multi-objective optimization problem		√	√	√								√					
248	MOEADDE_F1	Benchmark MOP for testing MOEA/D-DE		√		√					√								
249	MOEADDE_F2	Benchmark MOP for testing MOEA/D-DE		√		√					√								
250	MOEADDE_F3	Benchmark MOP for testing MOEA/D-DE		√		√					√								
251	MOEADDE_F4	Benchmark MOP for testing MOEA/D-DE		√		√					√								
252	MOEADDE_F5	Benchmark MOP for testing MOEA/D-DE		√		√					√								
253	MOEADDE_F6	Benchmark MOP for testing MOEA/D-DE		√		√					√								
254	MOEADDE_F7	Benchmark MOP for testing MOEA/D-DE		√		√					√								
255	MOEADDE_F8	Benchmark MOP for testing MOEA/D-DE		√		√					√								
256	MOEADDE_F9	Benchmark MOP for testing MOEA/D-DE		√		√					√								
257	MOEADM2M_F1	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
258	MOEADM2M_F2	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
259	MOEADM2M_F3	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
260	MOEADM2M_F4	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
261	MOEADM2M_F5	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
262	MOEADM2M_F6	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
263	MOEADM2M_F7	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
264	MOKP	The multi-objective knapsack problem		√	√				√		√								
265	MONRP	The multi-objective next release problem		√					√		√								
266	MOTSP	The multi-objective traveling salesman problem		√	√					√	√								
267	MPDMP	The multi-point distance minimization problem		√	√	√													
268	mQAP	The multi-objective quadratic assignment problem		√	√					√	√								
269	MW1	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
270	MW2	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
271	MW3	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
272	MW4	Constrained benchmark MOP proposed by		√	√	√					√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		Ma and Wang																	
273	MW5	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
274	MW6	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
275	MW7	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
276	MW8	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
277	MW9	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
278	MW10	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
279	MW11	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
280	MW12	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
281	MW13	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
282	MW14	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
283	NI_HS	Multitasking problem (Rosenbrock function + Rastrigin function)	√			√					√						√		
284	NI_MS	Multitasking problem (Griewank function + Weierstrass function)	√			√					√						√		
285	RMMEDA_F1	Benchmark MOP for testing RM-MEDA		√		√					√								
286	RMMEDA_F2	Benchmark MOP for testing RM-MEDA		√		√					√								
287	RMMEDA_F3	Benchmark MOP for testing RM-MEDA		√		√					√								
288	RMMEDA_F4	Benchmark MOP for testing RM-MEDA		√		√					√								
289	RMMEDA_F5	Benchmark MOP for testing RM-MEDA		√		√					√								
290	RMMEDA_F6	Benchmark MOP for testing RM-MEDA		√		√					√								
291	RMMEDA_F7	Benchmark MOP for testing RM-MEDA		√		√					√								
292	RMMEDA_F8	Benchmark MOP for testing RM-MEDA		√		√					√								
293	RMMEDA_F9	Benchmark MOP for testing RM-MEDA		√		√					√								
294	RMMEDA_F10	Benchmark MOP for testing RM-MEDA		√		√					√								
295	RWMOP1	Pressure vessal problem		√		√						√							
296	RWMOP2	Vibrating platform		√		√						√							
297	RWMOP3	Two bar truss design problem		√		√						√							
298	RWMOP4	Weldan beam design problem		√		√						√							
299	RWMOP5	Disc brake design problem		√		√						√							
300	RWMOP6	Speed reducer design problem		√		√						√							
301	RWMOP7	Gear train design problem		√		√						√							
302	RWMOP8	Car side impact design problem		√		√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
303	RWMOP9	Four bar plane truss		√		√						√							
304	RWMOP10	Two bar plane truss		√		√						√							
305	RWMOP11	Water resource management problem		√		√						√							
306	RWMOP12	Simply supported I-beam design		√		√						√							
307	RWMOP13	Gear box design		√		√						√							
308	RWMOP14	Multiple-disk clutch brake design problem		√		√						√							
309	RWMOP15	Spring design problem		√		√						√							
310	RWMOP16	Cantilever beam design problem		√		√						√							
311	RWMOP17	Bulk carriers design problem		√		√						√							
312	RWMOP18	Front rail design problem		√		√						√							
313	RWMOP19	Multi-product batch plant		√		√						√							
314	RWMOP20	Hydro-static thrust bearing design problem		√		√						√							
315	RWMOP21	Crash energy management for high-speed train		√		√						√							
316	RWMOP22	Haverly's pooling problem		√		√						√							
317	RWMOP23	Reactor network design		√		√						√							
318	RWMOP24	Heat exchanger network design		√		√						√							
319	RWMOP25	Process synthesis problem		√		√						√							
320	RWMOP26	Process sythesis and design problem		√		√						√							
321	RWMOP27	Process flow sheeting problem		√		√						√							
322	RWMOP28	Two reactor problem		√		√						√							
323	RWMOP29	Process synthesis problem		√		√						√							
324	RWMOP30	Synchronous pptimal pulse-width modulation of 3-level inverters		√		√						√							
325	RWMOP31	Synchronous pptimal pulse-width modulation of 5-level inverters		√		√						√							
326	RWMOP32	Synchronous pptimal pulse-width modulation of 7-level inverters		√		√						√							
327	RWMOP33	Synchronous pptimal pulse-width modulation of 9-level inverters		√		√						√							
328	RWMOP34	Synchronous pptimal pulse-width modulation of 11-level inverters		√		√						√							
329	RWMOP35	Synchronous pptimal pulse-width modulation of 13-level inverters		√		√						√							
330	RWMOP36	Optimal sizing of single phase distributed generation with reactive power support for phase balancing at main transformer/grid and active power loss		√		√						√							
331	RWMOP37	Optimal Sizing of Single Phase Distributed Generation with reactive power support for Phase Balancing at Main Transformer/Grid and reactive Power loss		√		√						√							
332	RWMOP38	Optimal sizing of single phase distributed generation with reactive power support for active and reactive power loss		√		√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
333	RWMOP39	Optimal sizing of single phase distributed generation with reactive power support for phase balancing at main transformer/grid and active and reactive power loss		√		√						√							
334	RWMOP40	Optimal power flow for minimizing active and reactive power loss		√		√						√							
335	RWMOP41	Optimal power flow for minimizing voltage deviation, active and reactive power loss		√		√						√							
336	RWMOP42	Optimal power flow for minimizing voltage deviation, and active power loss		√		√						√							
337	RWMOP43	Optimal power flow for minimizing fuel cost, and active power loss		√		√						√							
338	RWMOP44	Optimal power flow for minimizing fuel cost, active and reactive power loss		√		√						√							
339	RWMOP45	Optimal power flow for minimizing fuel cost, voltage deviation, and active power loss		√		√						√							
340	RWMOP46	Optimal power flow for minimizing fuel cost, voltage deviation, active and reactive power loss		√		√						√							
341	RWMOP47	Optimal droop setting for minimizing active and reactive power loss		√		√						√							
342	RWMOP48	Optimal droop setting for minimizing voltage deviation and active power loss		√		√						√							
343	RWMOP49	Optimal droop setting for minimizing voltage deviation, active, and reactive power loss		√		√						√							
344	RWMOP50	Power distribution system planning		√		√						√							
345	SMD1	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
346	SMD2	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
347	SMD3	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
348	SMD4	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
349	SMD5	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
350	SMD6	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
351	SMD7	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
352	SMD8	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
353	SMD9	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
354	SMD10	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
355	SMD11	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
356	SMD12	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
357	Sparse_CD	The community detection problem		√					√		√		√		√				
358	Sparse_CN	The critical node detection problem		√					√		√		√		√				
359	Sparse_FS	The feature selection problem		√					√		√		√		√				
360	Sparse_IS	The instance selection problem		√					√		√		√		√				
361	Sparse_KP	The sparse multi-objective knapsack problem		√	√				√		√								
362	Sparse_NN	The neural network training problem		√		√					√		√		√				
363	Sparse_PM	The pattern mining problem		√					√		√		√		√				
364	Sparse_PO	The portfolio optimization problem		√		√					√		√		√				
365	Sparse_SR	The sparse signal reconstruction problem		√		√					√		√		√				
366	SMMOP1	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
367	SMMOP2	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
368	SMMOP3	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
369	SMMOP4	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
370	SMMOP5	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
371	SMMOP6	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
372	SMMOP7	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
373	SMMOP8	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
374	SMOP1	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
375	SMOP2	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
376	SMOP3	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
377	SMOP4	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
378	SMOP5	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
379	SMOP6	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
380	SMOP7	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
381	SMOP8	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
382	SOP_F1	Sphere function	√			√							√						
383	SOP_F2	Schwefel's function 2.22	√			√							√						
384	SOP_F3	Schwefel's function 1.2	√			√							√						
385	SOP_F4	Schwefel's function 2.21	√			√							√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
386	SOP_F5	Generalized Rosenbrock's function	√			√							√						
387	SOP_F6	Step function	√			√							√						
388	SOP_F7	Quartic function with noise	√			√							√						
389	SOP_F8	Generalized Schwefel's function 2.26	√			√							√						
390	SOP_F9	Generalized Rastrigin's function	√			√							√						
391	SOP_F10	Ackley's function	√			√							√						
392	SOP_F11	Generalized Griewank's function	√			√							√						
393	SOP_F12	Generalized penalized function	√			√							√						
394	SOP_F13	Generalized penalized function	√			√							√						
395	SOP_F14	Shekel's foxholes function	√			√							√						
396	SOP_F15	Kowalik's function	√			√							√						
397	SOP_F16	Six-hump camel-back function	√			√							√						
398	SOP_F17	Branin function	√			√							√						
399	SOP_F18	Goldstein-price function	√			√							√						
400	SOP_F19	Hartman's family	√			√							√						
401	SOP_F20	Hartman's family	√			√							√						
402	SOP_F21	Shekel's family	√			√							√						
403	SOP_F22	Shekel's family	√			√							√						
404	SOP_F23	Shekel's family	√			√							√						
405	TP1	Test problem for robust multi-objective optimization		√		√					√								√
406	TP2	Test problem for robust multi-objective optimization		√		√					√								√
407	TP3	Test problem for robust multi-objective optimization		√		√					√								√
408	TP4	Test problem for robust multi-objective optimization		√		√					√								√
409	TP5	Test problem for robust multi-objective optimization		√		√					√								√
410	TP6	Test problem for robust multi-objective optimization		√		√					√								√
411	TP7	Test problem for robust multi-objective optimization		√		√					√								√
412	TP8	Test problem for robust multi-objective optimization		√		√					√								√
413	TP9	Test problem for robust multi-objective optimization		√		√					√								√
414	TP10	Test problem for robust multi-objective optimization		√		√					√	√							√
415	TREE1	The time-varying ratio error estimation problem		√		√					√	√	√						
416	TREE2	The time-varying ratio error estimation problem		√		√					√	√	√						
417	TREE3	The time-varying ratio error estimation problem		√		√					√	√	√						
418	TREE4	The time-varying ratio error estimation problem		√		√					√	√	√						
419	TREE5	The time-varying ratio error estimation problem		√		√					√	√	√						
420	TREE6	The time-varying ratio error estimation problem		√		√					√	√	√						
421	TSP	The traveling salesman problem	√							√	√								
422	UF1	Unconstrained benchmark MOP		√		√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
423	UF2	Unconstrained benchmark MOP		√		√					√								
424	UF3	Unconstrained benchmark MOP		√		√					√								
425	UF4	Unconstrained benchmark MOP		√		√					√								
426	UF5	Unconstrained benchmark MOP		√		√					√								
427	UF6	Unconstrained benchmark MOP		√		√					√								
428	UF7	Unconstrained benchmark MOP		√		√					√								
429	UF8	Unconstrained benchmark MOP		√		√					√								
430	UF9	Unconstrained benchmark MOP		√		√					√								
431	UF10	Unconstrained benchmark MOP		√		√					√								
432	VNT1	Benchmark MOP proposed by Viennet		√		√													
433	VNT2	Benchmark MOP proposed by Viennet		√		√													
434	VNT3	Benchmark MOP proposed by Viennet		√		√													
435	VNT4	Benchmark MOP proposed by Viennet		√		√						√							
436	WFG1	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
437	WFG2	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
438	WFG3	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
439	WFG4	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
440	WFG5	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
441	WFG6	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
442	WFG7	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
443	WFG8	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
444	WFG9	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
445	ZDT1	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
446	ZDT2	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
447	ZDT3	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
448	ZDT4	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
449	ZDT5	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√					√		√		√						
450	ZDT6	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
451	ZXH_CF1	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
452	ZXH_CF2	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
453	ZXH_CF3	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
454	ZXH_CF4	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
455	ZXH_CF5	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
456	ZXH_CF6	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
457	ZXH_CF7	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
458	ZXH_CF8	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
459	ZXH_CF9	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
460	ZXH_CF10	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
461	ZXH_CF11	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
462	ZXH_CF12	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
463	ZXH_CF13	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
464	ZXH_CF14	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
465	ZXH_CF15	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
466	ZXH_CF16	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							