PlatEMO

Evolutionary Multi-Objective Optimization Platform

User Manual 4.12

BIMK Group

April 14, 2025

[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* (*Prof. 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

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

where $\mathbf{x}$ 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;
- Function for calculating the gradients of objectives and constraints;
- 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:

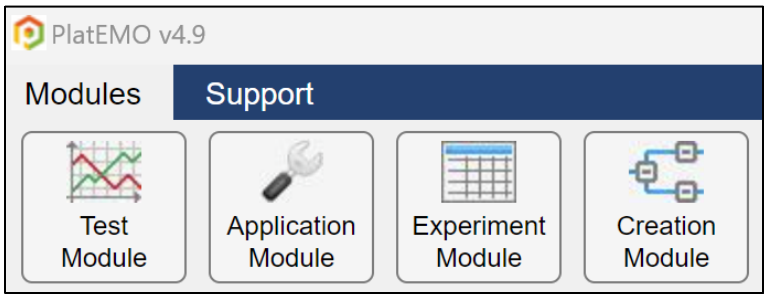
```
f1 = @(x)sum(x);
f2 = @(x)1-sum(x);
platemo('objFcn',f1,'conFcn',f2,'algorithm',@GA);
```

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 four 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), the experiment module is used to statistically analyze the performance of multiple algorithms on multiple benchmark problems (see *Functions of Experiment Module* for details), and the creation module is used to create new algorithms without writing code (see *Functions of Creation 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
`'run'`	Positive integer	`[]`	Current execution number
`'metName'`	Function handle or cell	`{}`	Names of metrics to calculate
`'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 `@DTLZ2` with 5 objectives via the algorithm `@NSGAII`:

```
platemo('algorithm',@NSGAII,'problem',@DTLZ2,'M',5);
```

- `'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 `@GA`:

```
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).
- `'run'` denotes the current execution number, which is involved in the name of saved files, differentiating the names of files saved for multiple executions of an algorithm on a problem (see *Collecting the Results* for details).
- `'metName'` denotes the names of metrics to calculate, whose value can be a string (a single metric) or a cell (multiple metrics). The metric values of saved

populations are calculated, and then are saved to a file or displayed in a figure (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 problems 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
`'gradFcn'`	Function handle	{}	Function for calculating the gradients of a solution on objectives and constraints Input: A decision vector Output 1: Jacobian matrix of objectives Output 2:Jacobian matrix of constraints
`'data'`	Any	{}	Data of the problem

`'once'`	Logical	0	Whether multiple solutions can be evaluated simultaneously

- `'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 a 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 a 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. The default `'decFcn'` limits each solution within the search space determined by `'lower'` and `'upper'`, while the following code defines a new `'decFcn'` to make 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:

```
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]);
```

- `'gradFcn'` denotes the function for calculating the gradients of a solution on objectives and constraints, whose value should be a function handle having one input and two outputs, where the input is a decision vector, the first output is the Jacobian matrix of objectives, and the second output is the Jacobian matrix of constraints. The default gradient function estimates the gradients via finite difference, while the following code defines a new `'objGradFcn'`:

```
function [oGrad,cGrad] = Grad(x)
```

```
    oGrad = [0,x(2:end);0,x(2:end)];
    cGrad = [0,x(2:end)-1/5];
end
```

Then, the following codes specify the gradient function:

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

Note that only a few algorithms use gradient functions.

- `'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);
```

- `'once'` indicates whether multiple solutions can be evaluated simultaneously, which should be a logical variable with default value of zero. When the value of `'once'` is set to 1, the inputs of `'evalFcn'`, `'decFcn'`, `'objFcn'`, and `'conFcn'` can be multiple decision vectors, i.e., evaluating multiple solutions simultaneously. Using matrix calculation or parallel calculation in functions can significantly improve the efficiency. For example, the following code updates the objective function with matrix calculation:

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

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

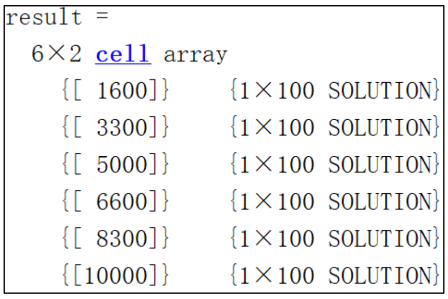
```
platemo('save',Value,…);
```

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. Moreover, the value of `run` can be explicitly specified by

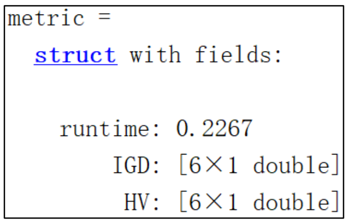
```
parfor i = 1 : 100
    platemo('save',Value,'run',i,…);
end
```

where `run` increases from 1 to 100. When multiple runs are performed in parallel, specifying the values of `run` can avoid the confusion or missing of file numbers.

Each 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.



```
result =
  6×2 cell array
    {[ 1600]}      {1×100 SOLUTION}
    {[ 3300]}      {1×100 SOLUTION}
    {[ 5000]}      {1×100 SOLUTION}
    {[ 6600]}      {1×100 SOLUTION}
    {[ 8300]}      {1×100 SOLUTION}
    {[10000]}      {1×100 SOLUTION}
```



```
metric =
  struct with fields:

    runtime: 0.2267
        IGD: [6×1 double]
         HV: [6×1 double]
```

Setting the parameter `'metName'` to specify the metrics to calculate, for example, the following code solves the problem `@DTLZ2` via the algorithm `@NSGAII` and saves the metric values of IGD and HV to a file:

```
platemo('algorithm',@NSGAII,'problem',@DTLZ2,...
'save',6,'metName',{'IGD','HV'});
```

where `'IGD'` and `'HV'` are the names of the metrics to calculate (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*. 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. Besides, the metric value of a single population can be calculated by

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

Also, the metric values can be automatically calculated and saved in the experiment module of the GUI.

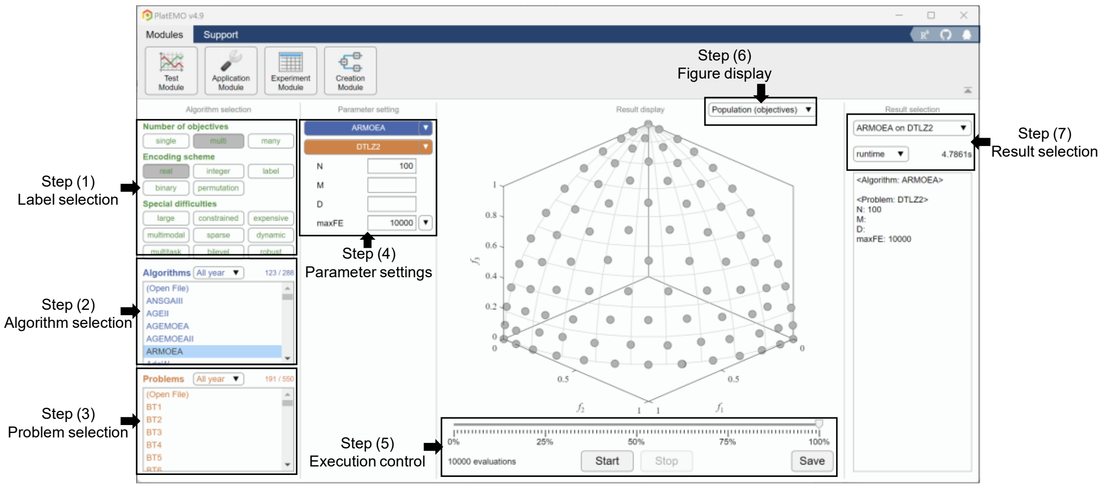
III. Using PlatEMO with GUI

A. 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.

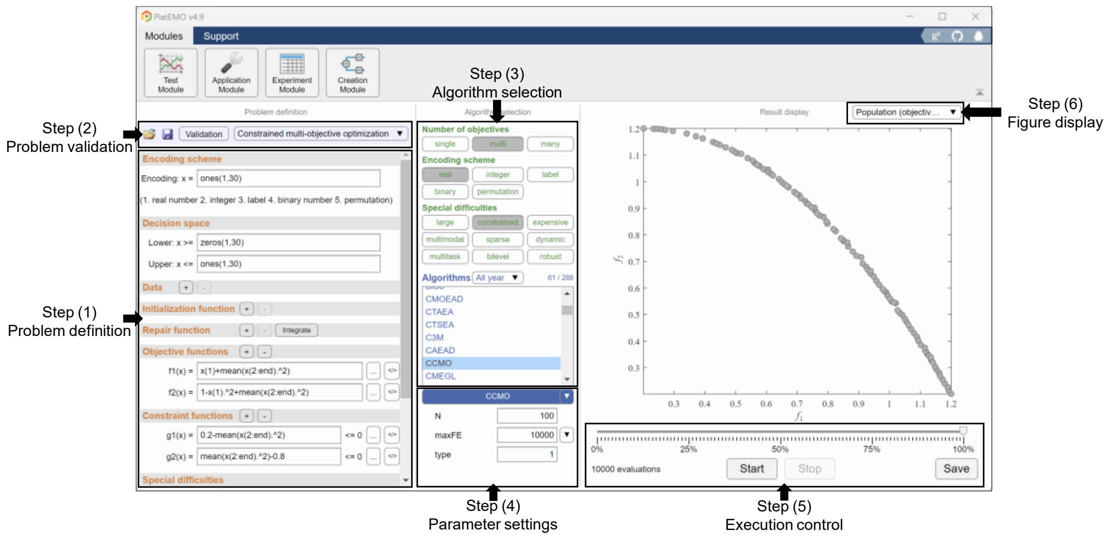


In this module, the performance investigation can be achieved by the following steps:

- 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. Application Module

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



Constrained multi-objective optimization

Encoding scheme

Encoding: x = ones(1,30)

(1. real number 2. integer 3. label 4. binary number 5. permutation)

Decision space

Lower: x >= zeros(1,30)

Upper: x <= ones(1,30)

Data

Initialization function

Repair function

Objective functions

f1(x) = x(1)+mean(x(2:end).^2)

f2(x) = 1-x(1).^2+mean(x(2:end).^2)

Constraint functions

g1(x) = 0.2-mean(x(2:end).^2) <= 0

g2(x) = mean(x(2:end).^2)-0.8 <= 0

Special difficulties

Number of objectives: single, multi, many

Encoding scheme: real, integer, label, binary, permutation

Special difficulties: large, constrained, expensive, multimodal, sparse, dynamic, multitask, bilevel, robust

Algorithms: All year — 61 / 288

CMOEAD, CTAEA, CTSEA, C3M, CAEAD, CCMO, CMEGL

CCMO

N: 100

maxFE: 10000

type: 1

Population (objectiv...

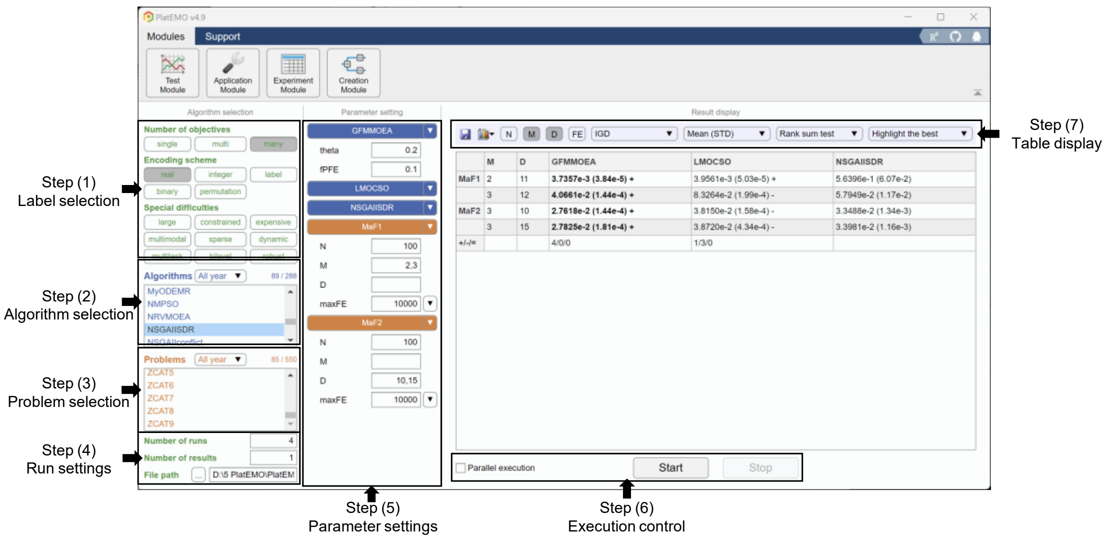
10000 evaluations

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. 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. The results generated in this module will be saved to MAT files (see *Collecting the Results* for details), and results will be loaded from existing files without execution.



Number of runs 4
Number of results 1
File path D:\5 PlatEMO\PlatEM

GFMMOEA
theta 0.2
fPFE 0.1
LMOCSO
NSGAIISDR
MaF1
N 100
M 2,3
D
maxFE 10000
MaF2
N 100
M
D 10,15
maxFE 10000

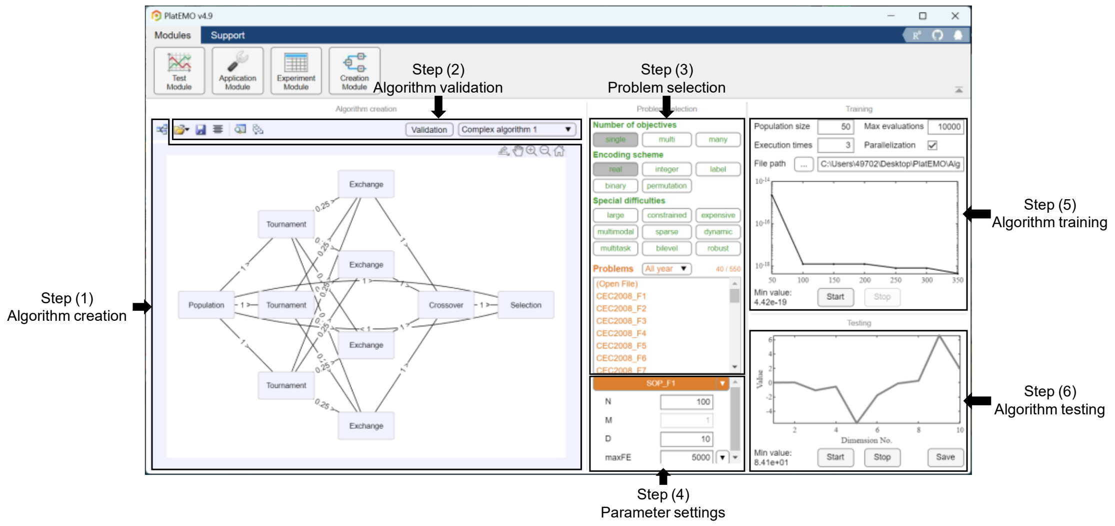
	M	D	GFMMOEA	LMOCSO	NSGAIISDR
MaF1	2	11	**3.7357e-3 (3.84e-5) +**	3.9561e-3 (5.03e-5) +	5.6396e-1 (6.07e-2)
	3	12	**4.0661e-2 (1.44e-4) +**	8.3264e-2 (1.99e-4) -	5.7949e-2 (1.17e-2)
MaF2	3	10	**2.7618e-2 (1.44e-4) +**	3.8150e-2 (1.58e-4) -	3.3488e-2 (1.34e-3)
	3	15	**2.7825e-2 (1.81e-4) +**	3.8720e-2 (4.34e-4) -	3.3981e-2 (1.16e-3)
+/-/=			4/0/0	1/3/0	

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. Here the parameters of benchmark problems can be vectors, which generates multiple test instances based on a single benchmark problem.
- 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. Creation Module

Users can press the menu button to switch to the creation module, which is used to create totally new algorithms and train them on benchmark problems.



In this module, new algorithms can be created and trained by the following steps:

- Step (1) Add new blocks by clicking on the button, add new connections by clicking on two blocks, change the layout by dragging blocks and connections. Blocks include population block, operator blocks, and selection blocks, where each block has some predefined hyperparameters and some parameters to train; connections indicate the transmission directions and ratios of solutions between blocks. An algorithm is regarded as a directed weighted cyclic graph with nodes of blocks and edges of connections, where the first node should be a population block, the algorithm should contain at least one node of operator block, all nodes should have predecessors and successors, all nodes should be reachable from any other, all cycles should contain at least one node of population block.
- Step (2) Save or load algorithms or blocks; generate source code of the algorithm; change the display style; automatically arrange the blocks; check the validity of the algorithm; select an algorithm template. After the algorithm is trained, users can generate source code of the algorithm and use it in other modules.
- Step (3) Select multiple labels to determine the type of problems (see *Labels of Algorithms, Problems, and Metrics* for details).
- Step (4) Set the parameters of the problem. Different problems may have different parameters, the details of which can be obtained by hovering over each parameter.
- Step (5) Train the parameters of all blocks of the algorithm on the selected problem. This process may be time-consuming, which may take several days for large number of blocks, number of variables, population size, and number of function evaluations.
- Step (6) Assess the performance of the trained algorithm on the selected problem.

E. 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
% <1995> <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.

In addition, each algorithm and benchmark problem can be tagged with a year label like `<2024>`, which enables them to be selected by year in the lists in the GUI.

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
`run`	Users	Current execution number
`metName`	Users	Names of metrics to calculate
`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
`starttime`	`NotTerminated()`	Used for runtime recording
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

```
classdef GA < ALGORITHM
% <1992><single><real/integer/label/binary/permutation><large/none><constrained/none>
% 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
`RouletteWheel Selection`	Roulette-wheel selection
`Tournament Selection`	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
`CalGrad`	Can	Calculate the gradients of a solution on objectives and constraints Input: A decision vector Output 1: Jacobian matrix of objectives Output 2: Jacobian matrix of constraints
`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
% <1999><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
```

```
12         function Setting(obj)
13             obj.M = 1;
14             if isempty(obj.D); obj.D = 30; end
15             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 CalGrad() estimates the gradients of objectives and 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 default 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 default 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
```

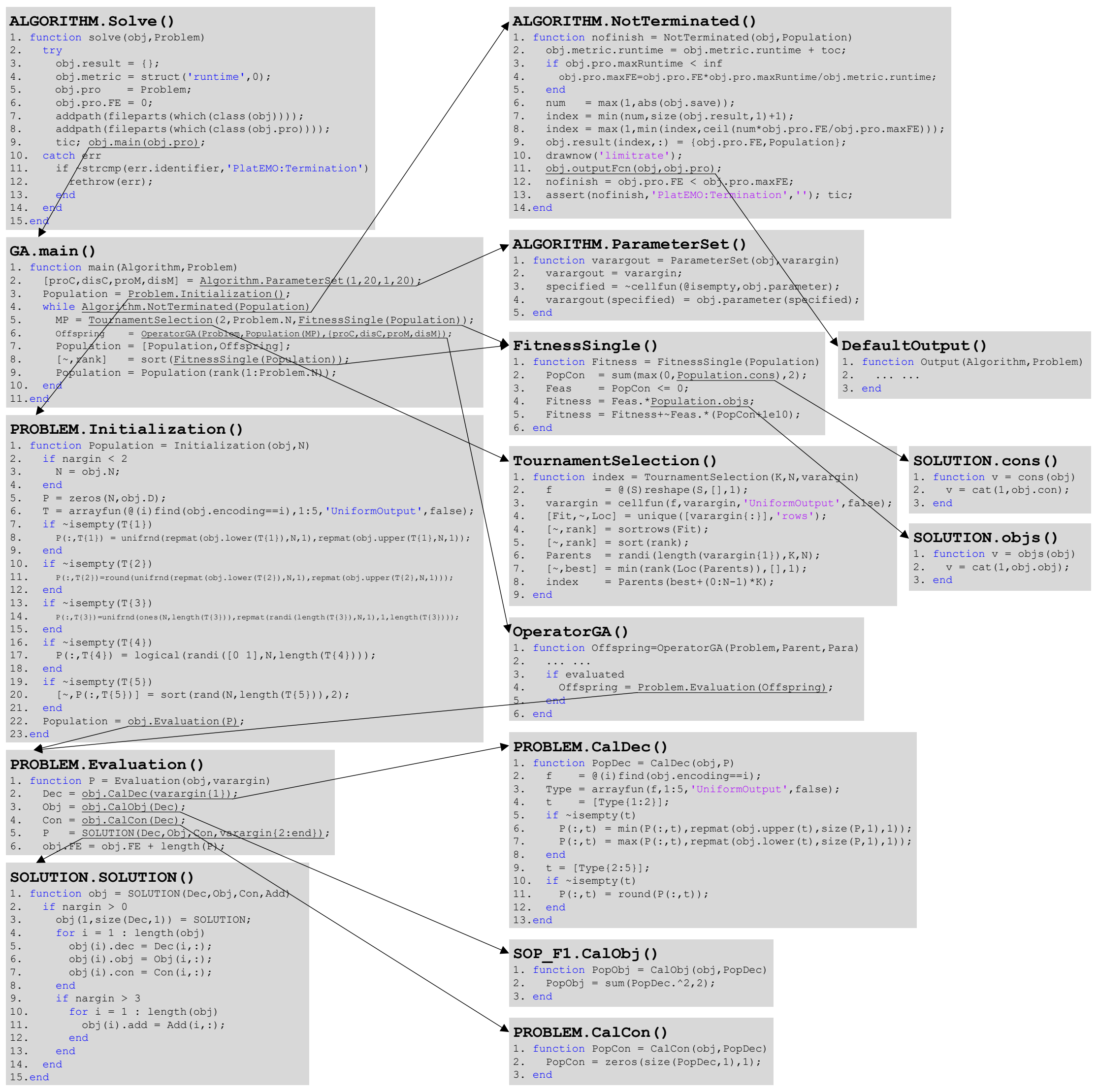
Note that `SOLUTION()` should be called only in the method `Evaluation()` of `PROBLEM` class.

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.



ALGORITHM.Solve()

```
function solve(obj,Problem)
  try
    obj.result = {};
    obj.metric = struct('runtime',0);
    obj.pro    = Problem;
    obj.pro.FE = 0;
    addpath(fileparts(which(class(obj))));
    addpath(fileparts(which(class(obj.pro))));
    tic; obj.main(obj.pro);
  catch err
    if ~strcmp(err.identifier,'PlatEMO:Termination')
      rethrow(err);
    end
  end
end
```

GA.main()

```
function main(Algorithm,Problem)
  [proC,disC,proM,disM] = Algorithm.ParameterSet(1,20,1,20);
  Population = Problem.Initialization();
  while Algorithm.NotTerminated(Population)
    MP = TournamentSelection(2,Problem.N,FitnessSingle(Population));
    Offspring    = OperatorGA(Problem,Population(MP),{proC,disC,proM,disM});
    Population = [Population,Offspring];
    [~,rank]   = sort(FitnessSingle(Population));
    Population = Population(rank(1:Problem.N));
  end
end
```

PROBLEM.Initialization()

```
function Population = Initialization(obj,N)
  if nargin < 2
    N = obj.N;
  end
  P = zeros(N,obj.D);
  T = arrayfun(@(i)find(obj.encoding==i),1:5,'UniformOutput',false);
  if ~isempty(T{1})
    P(:,T{1}) = unifrnd(repmat(obj.lower(T{1}),N,1),repmat(obj.upper(T{1},N,1));
  end
  if ~isempty(T{2})
    P(:,T{2})=round(unifrnd(repmat(obj.lower(T{2}),N,1),repmat(obj.upper(T{2},N,1)));
  end
  if ~isempty(T{3})
    P(:,T{3})=unifrnd(ones(N,length(T{3})),repmat(randi(length(T{3}),N,1),1,length(T{3})));
  end
  if ~isempty(T{4})
    P(:,T{4}) = logical(randi([0 1],N,length(T{4})));
  end
  if ~isempty(T{5})
    [~,P(:,T{5})] = sort(rand(N,length(T{5})),2);
  end
  Population = obj.Evaluation(P);
end
```

PROBLEM.Evaluation()

```
function P = Evaluation(obj,varargin)
  Dec = obj.CalDec(varargin{1});
  Obj = obj.CalObj(Dec);
  Con = obj.CalCon(Dec);
  P   = SOLUTION(Dec,Obj,Con,varargin{2:end});
  obj.FE = obj.FE + length(P);
```

SOLUTION.SOLUTION()

```
function obj = SOLUTION(Dec,Obj,Con,Add)
  if nargin > 0
    obj(1,size(Dec,1)) = SOLUTION;
    for i = 1 : length(obj)
      obj(i).dec = Dec(i,:);
      obj(i).obj = Obj(i,:);
      obj(i).con = Con(i,:);
    end
    if nargin > 3
      for i = 1 : length(obj)
        obj(i).add = Add(i,:);
      end
    end
  end
end
```

ALGORITHM.NotTerminated()

```
function nofinish = NotTerminated(obj,Population)
  obj.metric.runtime = obj.metric.runtime + toc;
  if obj.pro.maxRuntime < inf
    obj.pro.maxFE=obj.pro.FE*obj.pro.maxRuntime/obj.metric.runtime;
  end
  num   = max(1,abs(obj.save));
  index = min(num,size(obj.result,1)+1);
  index = max(1,min(index,ceil(num*obj.pro.FE/obj.pro.maxFE)));
  obj.result(index,:) = {obj.pro.FE,Population};
  drawnow('limitrate');
  obj.outputFcn(obj,obj.pro);
  nofinish = obj.pro.FE < obj.pro.maxFE;
  assert(nofinish,'PlatEMO:Termination',''); tic;
end
```

ALGORITHM.ParameterSet()

```
function varargout = ParameterSet(obj,varargin)
  varargout = varargin;
  specified = ~cellfun(@isempty,obj.parameter);
  varargout(specified) = obj.parameter(specified);
end
```

FitnessSingle()

```
function Fitness = FitnessSingle(Population)
  PopCon  = sum(max(0,Population.cons),2);
  Feas    = PopCon <= 0;
  Fitness = Feas.*Population.objs;
  Fitness = Fitness+~Feas.*(PopCon+1e10);
end
```

DefaultOutput()

```
function Output(Algorithm,Problem)
  ... ...
end
```

TournamentSelection()

```
function index = TournamentSelection(K,N,varargin)
  f        = @(S)reshape(S,[],1);
  varargin = cellfun(f,varargin,'UniformOutput',false);
  [Fit,~,Loc] = unique([varargin{:}],'rows');
  [~,rank] = sortrows(Fit);
  [~,rank] = sort(rank);
  Parents  = randi(length(varargin{1}),K,N);
  [~,best] = min(rank(Loc(Parents)),[],1);
  index    = Parents(best+(0:N-1)*K);
end
```

SOLUTION.cons()

```
function v = cons(obj)
  v = cat(1,obj.con);
end
```

SOLUTION.objs()

```
function v = objs(obj)
  v = cat(1,obj.obj);
end
```

OperatorGA()

```
function Offspring=OperatorGA(Problem,Parent,Para)
  ... ...
  if evaluated
    Offspring = Problem.Evaluation(Offspring);
  end
end
```

PROBLEM.CalDec()

```
function PopDec = CalDec(obj,P)
  f    = @(i)find(obj.encoding==i);
  Type = arrayfun(f,1:5,'UniformOutput',false);
  t    = [Type{1:2}];
  if ~isempty(t)
    P(:,t) = min(P(:,t),repmat(obj.upper(t),size(P,1),1));
    P(:,t) = max(P(:,t),repmat(obj.lower(t),size(P,1),1));
  end
  t = [Type{2:5}];
  if ~isempty(t)
    P(:,t) = round(P(:,t));
  end
end
```

SOP_F1.CalObj()

```
function PopObj = CalObj(obj,PopDec)
  PopObj = sum(PopDec.^2,2);
end
```

PROBLEM.CalCon()

```
function PopCon = CalCon(obj,PopDec)
  PopCon = zeros(size(PopDec,1),1);
end
```

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	AC-MMEA	Adaptive merging and coordinated offspring generation based multi-modal multi-objective evolutionary algorithm		√		√	√				√			√	√				
4	ACO	Ant colony optimization	√							√	√								
5	Adam	Adaptive moment estimation	√			√					√								
6	AdaW	Evolutionary algorithm with adaptive weights		√	√	√	√	√	√	√									
7	ADSAPSO	Adaptive dropout based surrogate-assisted particle swarm optimization		√	√	√	√						√						
8	AESSPSO	Adaptive exploration state-space particle swarm optimization	√			√	√				√	√							
9	AFSEA	Adjoint feature-selection-based evolutionary algorithm		√		√	√		√		√	√			√				
10	AGE-II	Approximation-guided evolutionary multi-objective algorithm II		√		√	√	√	√	√									
11	AGE-MOEA	Adaptive geometry estimation-based many-objective evolutionary algorithm		√	√	√	√	√	√	√		√							
12	AGE-MOEA-II	Adaptive geometry estimation-based many-objective evolutionary algorithm II		√	√	√	√	√	√	√		√							
13	AGSEA	Automated guiding vector selection-based evolutionary algorithm		√		√	√		√		√	√			√				
14	A-NSGA-III	Adaptive NSGA-III		√	√	√	√	√	√	√		√							
15	APSEA	Adaptive population sizing based evolutionary algorithm		√		√	√	√	√	√		√							
16	AR-MOEA	Adaptive reference points based multi-objective evolutionary algorithm		√	√	√	√	√	√	√		√							
17	AutoV	Automated design of variation operators	√	√		√	√				√	√							
18	AVG-SAEA	Adaptive variable grouping based surrogate-assisted evolutionary algorithm		√		√	√				√		√						
19	BCE-IBEA	Bi-criterion evolution based IBEA		√	√	√	√	√	√	√									
20	BCE-MOEA/D	Bi-criterion evolution based MOEA/D		√	√	√	√	√	√	√									
21	BFGS	A quasi-Newton method proposed by Broyden, Fletcher, Goldfarb, and Shanno	√			√					√								
22	BiCo	Bidirectional coevolution constrained multiobjective evolutionary algorithm		√		√	√	√	√	√		√							
23	BiGE	Bi-goal evolution			√	√	√	√	√	√									
24	BLEAQII	Bilevel evolutionary algorithm based on quadratic approximations II		√		√						√						√	

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
25	BL-SAEA	Bi-level surrogate modelling based evolutionary algorithm		√		√						√						√	
26	BSPGA	Binary space partition tree based genetic algorithm	√						√		√	√							
27	C3M	Constraint, multiobjective, multi-stage, multi-constraint evolutionary algorithm		√		√	√	√	√	√		√							
28	CAEAD	Dual-population evolutionary algorithm based on alternative evolution and degeneration		√		√	√	√	√	√		√							
29	CA-MOEA	Clustering based adaptive multi-objective evolutionary algorithm		√		√	√	√	√	√									
30	CCGDE3	Cooperative coevolution GDE3		√		√	√				√								
31	CCMO	Coevolutionary constrained multi-objective optimization framework		√		√	√	√	√	√		√							
32	c-DPEA	Constrained dual-population evolutionary algorithm		√		√	√	√	√	√		√							
33	CLIA	Evolutionary algorithm with cascade clustering and reference point incremental learning		√	√	√	√	√	√	√									
34	CMaDPPs	Constrained many-objective optimization with determinantal point processes		√	√	√	√	√	√	√		√							
35	CMA-ES	Covariance matrix adaptation evolution strategy	√			√	√				√	√							
36	CMDEIPCM	Constrained multiobjective differential evolution algorithm with an infeasible proportion control mechanism		√		√	√				√	√							
37	CMEGL	Constrained evolutionary multitasking with global and local auxiliary tasks		√		√	√	√	√	√		√							
38	CMME	Constrained many-objective evolutionary algorithm with enhanced mating and environmental selections		√		√	√	√	√	√		√							
39	CMMO	Coevolutionary multi-modal multi-objective optimization framework		√		√	√	√	√	√				√					
40	CMOCSO	Competitive and cooperative swarm optimization constrained multi-objective optimization algorithm		√		√					√	√							
41	CMODE-FTR	Constrained multiobjective differential evolution based on the fusion of two rankings		√		√	√					√							
42	CMOEA-CD	Constraint-Pareto dominance and diversity enhancement strategy based CMOEA		√	√	√	√	√	√	√		√							
43	C-MOEA/D	Constraint-MOEA/D		√	√	√	√	√	√	√		√							
44	CMOEA-MS	Constrained multiobjective evolutionary algorithm with multiple stages		√		√	√	√	√	√		√							
45	CMOEA-MSG	Multi-stage constrained multi-objective evolutionary algorithm		√		√	√					√							
46	CMOEMT	Constrained multi-objective optimization based on evolutionary multitasking optimization		√		√						√							
47	CMOES	Constrained multi-objective optimization based on even search		√		√	√	√	√	√		√							
48	CMOPSO	Competitive mechanism based multi-objective particle swarm optimizer		√		√	√												
49	CMOQLMT	Constrained multi-objective optimization based on Q-learning and multitasking		√		√						√							
50	CMOSMA	Constrained multi-objective evolutionary algorithm with self-organizing map		√	√	√	√					√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
51	CNSDE/DVC	Constrained nondominated sorting differential evolution based on decision variable classification		√		√	√												√
52	CoMMEA	Coevolutionary multimodal multi-objective evolutionary algorithm		√		√	√	√	√	√				√					
53	CPS-MOEA	Classification and Pareto domination based multi-objective evolutionary		√		√	√						√						
54	CSEA	Classification based surrogate-assisted evolutionary algorithm		√	√	√							√						
55	CSEMT	Constraints separation based evolutionary multitasking		√		√	√	√	√	√		√							
56	CSO	Competitive swarm optimizer	√			√	√				√	√							
57	C-TAEA	Two-archive evolutionary algorithm for constrained MOPs		√	√	√	√	√	√	√		√							
58	C-TSEA	Constrained two-stage evolutionary algorithm		√	√	√	√	√	√	√		√							
59	DAEA	Duplication analysis based evolutionary algorithm		√					√										
60	DBEMTO	Double-balanced evolutionary multi-task optimization		√		√	√	√	√	√		√							
61	DCNSGA-III	Dynamic constrained NSGA-III		√	√	√	√	√	√	√		√							
62	DE	Differential evolution	√			√	√				√	√							
63	DEA-GNG	Decomposition based evolutionary algorithm guided by growing neural gas		√	√	√	√	√	√	√									
64	DGEA	Direction guided evolutionary algorithm		√	√	√	√				√								
65	DirHV-EI	Expected direction-based hypervolume improvement		√	√	√	√						√						
66	DISK	Distribution-based Kriging-assisted evolutionary algorithm		√	√	√	√						√						
67	DISKplus	Distribution-based Kriging-assisted constrained evolutionary algorithm		√	√	√	√					√	√						
68	DKCA	Dynamic knowledge-guided coevolutionary algorithm		√		√			√		√	√			√				
69	DMOEA-eC	Decomposition-based multi-objective evolutionary algorithm with the e-constraint framework		√		√	√	√	√	√									
70	dMOPSO	MOPSO based on decomposition		√		√	√												
71	DN-NSGA-II	Decision space based niching NSGA-II		√		√	√							√					
72	DNSGA-II	Dynamic NSGA-II		√		√	√	√	√	√						√			
73	DOA	Dandelion optimization algorithm	√			√	√				√	√							
74	DPCPRA	Dual-population with dynamic constraint processing and resource allocating		√		√	√	√	√	√		√							
75	DP-PPS	Tri-population based push and pull search		√		√						√							
76	DPVAPS	Dual-population with variable auxiliary population size		√		√	√				√	√							
77	DRLOS-EMCMO	EMCMO with deep reinforcement learning-assisted operator selection		√		√	√	√	√	√		√							
78	DRL-SAEA	Deep reinforcement learning-based expensive constrained evolutionary algorithm		√		√						√	√						
79	DSPCMDE	Dynamic selection preference-assisted constrained multiobjective differential evolution		√		√	√					√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
80	DSSEA	Dynamic subspace search-based evolutionary algorithm		√	√	√	√				√	√							
81	DVCEA	Decision variables classification-based evolutionary algorithm		√	√	√	√				√	√							
82	DWU	Dominance-weighted uniformity multi-objective evolutionary algorithm		√		√	√	√	√	√									
83	EAG-MOEA/D	External archive guided MOEA/D		√		√	√	√	√	√									
84	ECPO	Electric charged particles optimization	√			√	√				√	√							
85	EDN-ARMOEA	Efficient dropout neural network based AR-MOEA		√	√	√	√						√						
86	EFR-RR	Ensemble fitness ranking with a ranking restriction scheme		√	√	√	√	√	√	√									
87	EGO	Efficient global optimization	√			√	√						√						
88	EIM-EGO	Expected improvement matrix based efficient global optimization		√		√	√						√						
89	EMCMMS	Evolutionary multitasking with a cooperative multistep mutation strategy		√		√	√	√	√	√		√							
90	EMCMO	Evolutionary multitasking-based constrained multiobjective optimization		√		√	√	√	√	√		√							
91	EMMOEA	Expensive multi-/many-objective evolutionary algorithm		√		√	√						√						
92	e-MOEA	Epsilon multi-objective evolutionary algorithm		√	√	√	√	√	√	√									
93	EMOSKT	Evolutionary multi-objective optimization with sparsity knowledge transfer		√		√			√		√	√			√		√		
94	EMyO/C	Evolutionary many-objective optimization algorithm with clustering-based		√	√	√	√												
95	ENS-MOEA/D	Ensemble of different neighborhood sizes based MOEA/D		√	√	√	√												
96	ESBCEO	Bayesian co-evolutionary optimization based entropy search		√		√							√						
97	FDV	Fuzzy decision variable framework with various internal optimizers		√	√	√	√				√								
98	FEP	Fast evolutionary programming	√			√	√				√	√							
99	FLEA	Fast sampling based evolutionary algorithm		√	√	√					√								
100	FRCG	Fletcher-Reeves conjugate gradient	√			√					√								
101	FRCGM	Fletcher-Reeves conjugate gradient (for multi-objective optimization)		√	√	√					√	√							
102	FROFI	Feasibility rule with the incorporation of objective function information	√			√	√				√	√							
103	GA	Genetic algorithm	√			√	√	√	√	√	√	√							
104	GCNMOEA	Graph convolutional network based multi-objective evolutionary algorithm		√		√	√												
105	GDE3	Generalized differential evolution 3		√		√	√					√							
106	GFM-MOEA	Generic front modeling based multi-objective evolutionary algorithm		√	√	√	√	√	√	√									
107	GLMO	Grouped and linked mutation operator algorithm		√		√	√				√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
108	g-NSGA-II	g-dominance based NSGA-II		√		√	√	√	√	√									
109	GPSO	Gradient based particle swarm optimization algorithm	√			√					√	√							
110	GPSOM	Gradient based particle swarm optimization algorithm (for multi-objective optimization)		√	√	√					√	√							
111	GrEA	Grid-based evolutionary algorithm			√	√	√	√	√	√									
112	GWASF-GA	Global weighting achievement scalarizing function genetic algorithm		√		√	√	√	√	√									
113	GWO	Grey wolf optimizer	√			√	√				√	√							
114	HEA	Hyper-dominance based evolutionary algorithm		√	√	√			√	√									
115	HeE-MOEA	Multiobjective evolutionary algorithm with heterogeneous ensemble based infill criterion		√		√	√						√						
116	HHC-MMEA	Hybrid hierarchical clustering based multi-modal multi-objective evolutionary algorithm		√		√					√			√	√				
117	hpaEA	Hyperplane assisted evolutionary algorithm		√	√	√	√	√	√	√									
118	HREA	Hierarchy ranking based evolutionary algorithm		√		√	√							√					
119	HypE	Hypervolume estimation algorithm		√	√	√	√	√	√	√									
120	IBEA	Indicator-based evolutionary algorithm		√	√	√	√	√	√	√									
121	ICMA	Indicator based constrained multi-objective algorithm		√		√	√					√							
122	I-DBEA	Improved decomposition-based evolutionary algorithm		√	√	√	√	√	√	√		√							
123	IM-C-MOEA/D	Inverse modeling constrained MOEA/D		√		√	√				√	√							
124	IM-MOEA	Inverse modeling based multiobjective evolutionary algorithm		√		√	√				√								
125	IM-MOEA/D	Inverse modeling MOEA/D		√		√	√				√								
126	IMODE	Improved multi-operator differential evolution	√			√	√				√	√							
127	IMTCMO	Improved evolutionary multitasking-based CMOEA		√		√	√	√	√	√		√							
128	IMTCMO_BS	Improved evolutionary multitasking-based CMOEA with bidirectional sampling		√	√	√	√	√	√	√		√							
129	I-SIBEA	Interactive simple indicator-based evolutionary algorithm		√		√	√	√	√	√									
130	Izui	An aggregative gradient based multi-objective optimizer proposed by Izui et al.		√	√	√					√	√							
131	KMA	Komodo mlipir algorithm	√			√	√				√	√							
132	KnEA	Knee point driven evolutionary algorithm			√	√	√	√	√	√		√							
133	K-RVEA	Surrogate-assisted RVEA		√	√	√	√						√						
134	KTA2	Kriging-assisted Two_Arch2		√	√	√	√						√						
135	KTS	Kriging-assisted evolutionary algorithm with two search modes		√	√		√					√	√						
136	L2SMEA	Linear subspace surrogate modeling assisted evolutionary algorithm	√			√							√						
137	LCMEA	Large-scale constrained multi-objective evolutionary algorithm		√		√					√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
138	LCSA	Linear combination-based search algorithm		√	√	√	√				√								
139	LDS-AF	Low-dimensional surrogate aggregation function		√		√	√				√		√						
140	LERD	Large-scale evolutionary algorithm with reformulated decision variable analysis		√	√	√					√								
141	LMEA	Evolutionary algorithm for large-scale many-objective optimization		√	√	√	√				√								
142	LMOCSO	Large-scale multi-objective competitive swarm optimization algorithm		√	√	√	√				√	√							
143	LMOEA-DS	Large-scale evolutionary multi-objective optimization assisted by directed sampling		√		√	√				√								
144	LMPFE	Evolutionary algorithm with local model based Pareto front estimation		√	√	√	√	√	√	√									
145	LRMOEA	Large-scale robust multi-objective evolutionary algorithm		√		√			√		√	√			√				√
146	LSMOF	Large-scale multi-objective optimization framework with NSGA-II		√		√	√				√								
147	MaOEA-CSS	Many-objective evolutionary algorithms based on coordinated selection		√	√	√	√	√	√	√									
148	MaOEA-DDFC	Many-objective evolutionary algorithm based on directional diversity and favorable convergence		√	√	√	√	√	√	√									
149	MaOEA/IGD	IGD based many-objective evolutionary algorithm			√	√	√	√	√	√									
150	MaOEA/IT	Many-objective evolutionary algorithms based on an independent two-stage		√	√	√	√					√							
151	MaOEA-R&D	Many-objective evolutionary algorithm based on objective space reduction			√	√	√	√	√	√									
152	MCCMO	Multi-population coevolutionary constrained multi-objective optimization		√		√	√	√	√	√		√							
153	MCEA/D	Multiple classifiers-assisted evolutionary algorithm based on decomposition		√	√	√	√						√						
154	MFEA	Multifactorial evolutionary algorithm	√			√	√	√	√	√	√						√		
155	MFEA-II	Multifactorial evolutionary algorithm II	√			√	√	√	√	√	√						√		
156	MFFS	Multiform feature selection		√					√										
157	MFO-SPEA2	Multiform optimization framework based on SPEA2		√		√	√	√	√	√		√							
158	MGCEA	Multi-granularity clustering based evolutionary algorithm		√		√			√		√	√			√				
159	MGO	Mountain gazelle optimizer	√			√	√				√	√							
160	MGSAEA	Multigranularity surrogate-assisted constrained evolutionary algorithm		√		√						√	√						
161	MMEAPSL	Multimodal multi-objective evolutionary algorithm assisted by Pareto set learning		√		√	√	√	√	√				√					
162	MMEA-WI	Weighted indicator-based evolutionary algorithm for multimodal multi-objective optimization		√		√	√							√					
163	MMOPSO	MOPSO with multiple search strategies		√		√	√												
164	MO_Ring_ PSO_SCD	Multiobjective PSO using ring topology and special crowding distance		√		√	√							√					
165	MOBCA	Multi-objective besiege and conquer algorithm		√		√	√												

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
166	MOCell	Cellular genetic algorithm		√		√	√	√	√	√		√							
167	MOCGDE	Multi-objective conjugate gradient and differential evolution algorithm		√	√	√					√	√							
168	MO-CMA	Multi-objective covariance matrix adaptation evolution strategy		√		√	√												
169	MOEA/CKF	Multi-objective evolutionary algorithm based on cross-scale knowledge fusion		√		√			√		√	√			√				
170	MOEA/D	Multiobjective evolutionary algorithm based on decomposition		√	√	√	√	√	√	√									
171	MOEA/D-2WA	MOEA/D with two-type weight vector adjustments		√	√	√	√	√	√	√		√							
172	MOEA/D-AWA	MOEA/D with adaptive weight adjustment		√	√	√	√	√	√	√									
173	MOEA/D-CMA	MOEA/D with covariance matrix adaptation evolution strategy		√	√	√	√												
174	MOEA/D-CMT	MOEA/D with competitive multitasking		√		√						√							
175	MOEA/DD	Many-objective evolutionary algorithm based on dominance and decomposition		√	√	√	√	√	√	√		√							
176	MOEA/D-DAE	MOEA/D with detect-and-escape strategy		√		√	√	√	√	√		√							
177	MOEA/D-DCWV	MOEA/D with distribution control of weight vector set		√	√	√	√	√	√	√									
178	MOEA/D-DE	MOEA/D based on differential evolution		√	√	√	√												
179	MOEA/D-DQN	MOEA/D based on deep Q-network		√	√	√	√												
180	MOEA/D-DRA	MOEA/D with dynamical resource allocation		√	√	√	√												
181	MOEA/D-DU	MOEA/D with a distance based updating strategy		√	√	√	√	√	√	√									
182	MOEA/D-DYTS	MOEA/D with dynamic Thompson sampling		√	√	√	√												
183	MOEA/D-EGO	MOEA/D with efficient global optimization		√		√	√						√						
184	MOEA/D-FRRMAB	MOEA/D with fitness-rate-rank-based multiarmed bandit		√	√	√	√												
185	MOEA/D-M2M	MOEA/D based on MOP to MOP		√		√	√												
186	MOEA/D-MRDL	MOEA/D with maximum relative diversity loss		√		√	√												
187	MOEA/D-PaS	MOEA/D with Pareto adaptive scalarizing approximation		√	√	√	√												
188	MOEA/D-PFE	MOEA/D with Pareto front estimation		√	√	√	√	√	√	√									
189	MOEA/D-STM	MOEA/D with stable matching		√	√	√	√												
190	MOEA/D-UR	MOEA/D with update when required		√	√	√	√	√	√	√									
191	MOEA/D-URAW	MOEA/D with uniform randomly adaptive weights		√	√	√	√	√	√	√									
192	MOEA/DVA	Multi-objective evolutionary algorithm based on decision variable		√		√	√				√								
193	MOEA/D-VOV	MOEA/D with virtual objective vectors		√	√	√	√	√	√	√									
194	MOEA/IGD-NS	Multi-objective evolutionary algorithm based on an enhanced IGD		√		√	√	√	√	√									

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
195	MOEA-NZD	Multi-objective evolutionary algorithm with nonzero detection		√	√	√					√	√			√				
196	MOEA-PC	Multiobjective evolutionary algorithm based on polar coordinates		√		√	√												
197	MOEA/PSL	Multi-objective evolutionary algorithm based on Pareto optimal subspace		√		√	√		√		√	√			√				
198	MOEA-RE	Multi-objective evolutionary algorithm with robustness enhancement		√		√	√	√	√	√									√
199	MO-EGS	Multi-objective evolutionary gradient search		√		√					√								
200	MO-L2SMEA	Multi-objective linear subspace surrogate modeling assisted evolutionary algorithm		√		√					√		√						
201	MOMBI-II	Many objective metaheuristic based on the R2 indicator II		√	√	√	√	√	√	√									
202	MO-MFEA	Multi-objective multifactorial evolutionary algorithm		√		√	√	√	√	√		√					√		
203	MO-MFEA-II	Multi-objective multifactorial evolutionary algorithm II		√		√	√	√	√	√		√					√		
204	MOMFEA-SADE	Multi-objective multifactorial evolutionary algorithm with subspace alignment and adaptive differential evolution		√		√	√	√	√	√		√					√		
205	MOPSO	Multi-objective particle swarm optimization		√		√	√												
206	MOPSO-CD	MOPSO with crowding distance		√		√	√												
207	MOSD	Multiobjective steepest descent		√		√					√	√							
208	M-PAES	Memetic algorithm with Pareto archived evolution strategy		√		√	√												
209	MP-MMEA	Multi-population multi-modal multi-objective evolutionary algorithm		√		√	√				√			√	√				
210	MPSO/D	Multi-objective particle swarm optimization algorithm based on decomposition		√	√	√	√												
211	MSCEA	Multi-stage constrained multi-objective evolutionary algorithm		√		√	√	√	√	√		√							
212	MSCMO	Multi-stage constrained multi-objective evolutionary algorithm		√		√	√	√	√	√		√							
213	MSEA	Multi-stage multi-objective evolutionary algorithm		√		√	√	√	√	√									
214	MSKEA	Multi-stage knowledge-guided evolutionary algorithm		√		√	√		√		√	√			√				
215	MSOPS-II	Multiple single objective Pareto sampling II		√	√	√	√					√							
216	MTCMO	Multitasking constrained multi-objective optimization		√		√	√	√	√	√		√							
217	MTDE-MKTA	Multitasking differential evolution with multiple knowledge types and transfer adaptation		√		√	√	√	√	√		√					√		
218	MTEA/D-DN	Multiobjective multitask evolutionary algorithm based on decomposition with dual neighborhoods		√		√	√	√	√	√		√					√		
219	MTS	Multiple trajectory search		√		√	√												
220	MultiObjective EGO	Multi-objective efficient global optimization		√		√	√					√	√						
221	MVPA	Most valuable player algorithm	√			√	√				√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
222	MyO-DEMR	Many-objective differential evolution with mutation restriction		√	√	√	√												
223	NBLEA	Nested bilevel evolutionary algorithm		√		√						√						√	
224	NelderMead	The Nelder-Mead algorithm	√			√													
225	NMPSO	Novel multi-objective particle swarm optimization		√	√	√	√												
226	NNDREA-MO	Evolutionary algorithm with neural network-based dimensionality reduction (multi-objective)		√					√		√	√			√				
227	NNDREA-SO	Evolutionary algorithm with neural network-based dimensionality reduction (single-objective)		√					√		√	√			√				
228	NNIA	Nondominated neighbor immune algorithm		√		√	√	√	√	√									
229	NRV-MOEA	Adaptive normal reference vector-based multi- and many-objective evolutionary algorithm		√	√	√	√	√	√	√									
230	NSBiDiCo	Non-dominated sorting bidirectional differential coevolution algorithm		√		√	√	√	√	√		√							
231	NSGA-II	Nondominated sorting genetic algorithm II		√		√	√	√	√	√		√							
232	NSGA-II+ARSBX	NSGA-II with adaptive rotation based simulated binary crossover		√		√	√					√							
233	NSGA-II-conflict	NSGA-II with conflict-based partitioning strategy			√	√	√	√	√	√									
234	NSGA-II-DTI	NSGA-II of Deb's type I robust version		√		√	√	√	√	√		√							√
235	NSGA-III	Nondominated sorting genetic algorithm III		√	√	√	√	√	√	√		√							
236	NSGAIII-EHVI	NSGA-III with expected hypervolume improvement		√	√	√							√						
237	NSGA-II/SDR	NSGA-II with strengthened dominance relation			√	√	√	√	√	√									
238	NSLS	Multiobjective optimization framework based on nondominated sorting and local search		√		√	√												
239	NUCEA	Non-uniform clustering based evolutionary algorithm		√		√			√		√	√			√				
240	OFA	Optimal foraging algorithm	√			√	√				√	√							
241	one-by-one EA	Many-objective evolutionary algorithm using a one-by-one selection		√	√	√	√	√	√	√									
242	OSP-NSDE	Non-dominated sorting differential evolution with prediction in the objective space		√		√	√												
243	ParEGO	Efficient global optimization for Pareto optimization		√		√	√						√						
244	PB-NSGA-III	NSGA-III based on Pareto based bi-indicator infill sampling criterion		√	√	√	√						√						
245	PB-RVEA	RVEA based on Pareto based bi-indicator infill sampling criterion		√	√	√	√						√						
246	PC-SAEA	Pairwise comparison based surrogate-assisted evolutionary algorithm		√	√								√						
247	PeEA	Pareto front shape estimation based evolutionary algorithm		√	√	√	√	√	√	√									
248	PESA-II	Pareto envelope-based selection algorithm II		√		√	√	√	√	√									
249	PICEA-g	Preference-inspired coevolutionary algorithm with goals		√	√	√	√	√	√	√									
250	PIEA	Performance indicator-based evolutionary algorithm		√	√	√							√						
251	PIMD	Probability and mapping crowding distance		√	√	√							√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
252	PM-MOEA	Pattern mining based multi-objective evolutionary algorithm		√		√	√		√		√	√			√				
253	POCEA	Paired offspring generation based constrained evolutionary algorithm		√		√	√				√	√							
254	PPS	Push and pull search algorithm		√	√	√	√					√							
255	PRDH	Problem reformulation and duplication handling		√					√										
256	PREA	Promising-region based EMO algorithm		√	√	√	√	√	√	√									
257	PSO	Particle swarm optimization	√			√	√				√	√							
258	REMO	Expensive multiobjective optimization by relation learning and prediction		√	√	√							√						
259	RGA-M1-2	Real-coded genetic algorithm with framework M1-2		√		√						√	√						
260	RGA-M2-2	Real-coded genetic algorithm with framework M2-2		√		√						√	√						
261	RM-MEDA	Regularity model-based multiobjective estimation of distribution		√		√	√												
262	RMOEA/DVA	Robust multi-objective evolutionary algorithm with decision variable assortment		√		√	√												√
263	RMSProp	Root mean square propagation	√			√					√								
264	r-NSGA-II	r-dominance based NSGA-II		√		√	√	√	√	√									
265	RPD-NSGA-II	Reference point dominance-based NSGA-II		√	√	√	√	√	√	√									
266	RPEA	Reference points-based evolutionary algorithm			√	√	√	√	√	√									
267	RSEA	Radial space division based evolutionary algorithm		√	√	√	√	√	√	√									
268	RVEA	Reference vector guided evolutionary algorithm		√	√	√	√	√	√	√		√							
269	RVEAa	RVEA embedded with the reference vector regeneration strategy			√	√	√	√	√	√									
270	RVEA-iGNG	RVEA based on improved growing neural gas		√	√	√	√	√	√	√									
271	S3-CMA-ES	Scalable small subpopulations based covariance matrix adaptation		√	√	√	√				√								
272	SA	Simulated annealing	√			√	√				√	√							
273	SACC-EAM-II	Surrogate-assisted cooperative co-evolutionary algorithm of Minamo	√			√	√						√						
274	SACOSO	Surrogate-assisted cooperative swarm optimization	√			√	√				√		√						
275	SADE-AMSS	Surrogate-assisted differential evolution with adaptive multi-subspace search	√			√	√						√						
276	SADE-ATDSC	Surrogate-assisted differential evolution with adaptation of training data selection criterion	√			√	√						√						
277	SADE-Sammon	Sammon mapping assisted differential evolution	√			√	√						√						
278	SAMSO	Multiswarm-assisted expensive optimization	√			√	√				√		√						
279	SAPO	Surrogate-assisted partial optimization	√			√	√					√	√						
280	S-CDAS	Self-controlling dominance area of solutions			√	√	√	√	√	√									
281	SCEA	Sparsity clustering basec evolutionary algorithm		√		√			√		√	√			√				
282	SD	Steepest descent	√			√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
283	S-ECSO	Enhanced competitive swarm optimizer for sparse optimization		√		√					√				√				
284	SFADE	Scalarization function approximation based differential evolution algorithm		√	√	√	√						√						
285	SGEA	Steady-state and generational evolutionary algorithm		√		√	√	√	√	√		√				√			
286	SGECF	Sparsity-guided elitism co-evolutionary framework		√		√			√		√	√			√				
287	SHADE	Success-history based adaptive differential evolution	√			√	√				√	√							
288	SIBEA	Simple indicator-based evolutionary algorithm		√		√	√	√	√	√									
289	SIBEA-kEMOSS	SIBEA with minimum objective subset of size k with minimum error			√	√	√	√	√	√									
290	SLMEA	Super-large-scale multi-objective evolutionary algorithm		√		√	√		√		√	√			√				
291	SMEA	Self-organizing multiobjective evolutionary algorithm		√		√	√												
292	SMOA	Supervised multi-objective optimization algorithm		√		√							√						
293	SMPSO	Speed-constrained multi-objective particle swarm optimization		√		√	√												
294	SMS-EGO	S metric selection based efficient global optimization		√		√	√						√						
295	SMS-EMOA	S metric selection based evolutionary multiobjective optimization		√		√	√	√	√	√									
296	S-NSGA-II	Sparse NSGA-II		√		√					√	√			√				
297	SparseEA	Evolutionary algorithm for sparse multi-objective optimization problems		√		√	√		√		√	√			√				
298	SparseEA2	Improved SparseEA		√		√	√		√		√	√			√				
299	SPEA2	Strength Pareto evolutionary algorithm 2		√		√	√	√	√	√									
300	SPEA2+SDE	SPEA2 with shift-based density estimation			√	√	√	√	√	√									
301	SPEA/R	Strength Pareto evolutionary algorithm based on reference direction		√	√	√	√	√	√	√									
302	SQP	Sequential quadratic programming	√			√					√	√							
303	SRA	Stochastic ranking algorithm			√	√	√	√	√	√									
304	SSCEA	Subspace segmentation based co-evolutionary algorithm		√	√	√	√												
305	SSDE	Self-organized surrogate-assisted differential evolution		√	√	√	√					√	√						
306	t-DEA	theta-dominance based evolutionary algorithm		√	√	√	√	√	√	√									
307	tDEA-CPBI	Theta-dominance based evolutionary algorithm with CPBI		√	√	√	√	√	√	√		√							
308	TELSO	Two-layer encoding learning swarm optimizer		√		√			√		√	√			√				
309	TiGE-2	Tri-Goal Evolution Framework for CMaOPs			√	√	√	√	√	√		√							
310	ToP	Two-phase framework with NSGA-II		√		√	√					√							
311	TPCMaO	Three-population based constrained many-objective co-evolutionary algorithm			√	√	√	√	√	√		√							
312	TriMOEA-	Multi-modal MOEA using two-archive and		√		√	√							√					

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
	TA&R	recombination strategies																	
313	TS-NSGA-II	Two-stage NSGA-II		√	√	√	√	√	√	√									
314	TS-SparseEA	Two-stage SparseEA		√		√			√		√	√			√				
315	TSTI	Two-stage evolutionary algorithm with three indicators		√		√	√	√	√	√		√							
316	Two_Arch2	Two-archive algorithm 2		√	√	√	√	√	√	√									
317	URCMO	Utilizing the relationship between constrained and unconstrained Pareto fronts for constrained multi-objective optimization		√		√	√					√							
318	VaEA	Vector angle based evolutionary algorithm		√	√	√	√	√	√	√									
319	WASF-GA	Weighting achievement scalarizing function genetic algorithm		√		√	√	√	√	√									
320	WOA	Whale optimization algorithm	√			√	√				√	√							
321	WOF	Weighted optimization framework		√		√	√				√								
322	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	BBOB_F1	Sphere function	√			√							√						
2	BBOB_F2	Ellipsoidal function	√			√							√						
3	BBOB_F3	Rastrigin function	√			√							√						
4	BBOB_F4	Buche-Rastrigin function	√			√							√						
5	BBOB_F5	Linear slope	√			√							√						
6	BBOB_F6	Attractive sector function	√			√							√						
7	BBOB_F7	Step ellipsoidal function	√			√							√						
8	BBOB_F8	Rosenbrock function	√			√							√						
9	BBOB_F9	Rotated Rosenbrock function	√			√							√						
10	BBOB_F10	Rotated ellipsoidal function	√			√							√						
11	BBOB_F11	Discus function	√			√							√						
12	BBOB_F12	Bent cigar function	√			√							√						
13	BBOB_F13	Sharp ridge function	√			√							√						
14	BBOB_F14	Different powers function	√			√							√						
15	BBOB_F15	Rastrigin function	√			√							√						
16	BBOB_F16	Weierstrass function	√			√							√						
17	BBOB_F17	Schaffers F7 function	√			√							√						
18	BBOB_F18	Moderately ill-conditioned Schaffers F7 function	√			√							√						
19	BBOB_F19	Composite Griewank-Rosenbrock function F8F2	√			√							√						
20	BBOB_F20	Schwefel function	√			√							√						
21	BBOB_F21	Gallagher's Gaussian 101-me peaks function	√			√							√						
22	BBOB_F22	Gallagher's Gaussian 21-hi peaks function	√			√							√						
23	BBOB_F23	Katsuura function	√			√							√						
24	BBOB_F24	Lunacek bi-Rastrigin function	√			√							√						
25	BT1	Benchmark MOP with bias feature		√		√					√								
26	BT2	Benchmark MOP with bias feature		√		√					√								
27	BT3	Benchmark MOP with bias feature		√		√					√								
28	BT4	Benchmark MOP with bias feature		√		√					√								
29	BT5	Benchmark MOP with bias feature		√		√					√								
30	BT6	Benchmark MOP with bias feature		√		√					√								
31	BT7	Benchmark MOP with bias feature		√		√					√								
32	BT8	Benchmark MOP with bias feature		√		√					√								
33	BT9	Benchmark MOP with bias feature		√		√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
34	C10MOP1	Neural architecture search on CIFAR-10		√		√					√		√						
35	C10MOP2	Neural architecture search on CIFAR-10		√		√					√		√						
36	C10MOP3	Neural architecture search on CIFAR-10		√		√					√		√						
37	C10MOP4	Neural architecture search on CIFAR-10		√		√					√		√						
38	C10MOP5	Neural architecture search on CIFAR-10		√		√					√		√						
39	C10MOP6	Neural architecture search on CIFAR-10		√		√					√		√						
40	C10MOP7	Neural architecture search on CIFAR-10		√		√					√		√						
41	C10MOP8	Neural architecture search on CIFAR-10		√		√					√		√						
42	C10MOP9	Neural architecture search on CIFAR-10		√		√					√		√						
43	CEC2008_F1	Shifted sphere function	√			√					√		√						
44	CEC2008_F2	Shifted Schwefel's function	√			√					√		√						
45	CEC2008_F3	Shifted Rosenbrock's function	√			√					√		√						
46	CEC2008_F4	Shifted Rastrign's function	√			√					√		√						
47	CEC2008_F5	Shifted Griewank's function	√			√					√		√						
48	CEC2008_F6	Shifted Ackley's function	√			√					√		√						
49	CEC2008_F7	FastFractal 'DoubleDip' function	√			√					√		√						
50	CEC2010_F1	CEC'2010 constrained optimization benchmark problem	√			√						√							
51	CEC2010_F2	CEC'2010 constrained optimization benchmark problem	√			√						√							
52	CEC2010_F3	CEC'2010 constrained optimization benchmark problem	√			√						√							
53	CEC2010_F4	CEC'2010 constrained optimization benchmark problem	√			√						√							
54	CEC2010_F5	CEC'2010 constrained optimization benchmark problem	√			√						√							
55	CEC2010_F6	CEC'2010 constrained optimization benchmark problem	√			√						√							
56	CEC2010_F7	CEC'2010 constrained optimization benchmark problem	√			√						√							
57	CEC2010_F8	CEC'2010 constrained optimization benchmark problem	√			√						√							
58	CEC2010_F9	CEC'2010 constrained optimization benchmark problem	√			√						√							
59	CEC2010_F10	CEC'2010 constrained optimization benchmark problem	√			√						√							
60	CEC2010_F11	CEC'2010 constrained optimization benchmark problem	√			√						√							
61	CEC2010_F12	CEC'2010 constrained optimization benchmark problem	√			√						√							
62	CEC2010_F13	CEC'2010 constrained optimization benchmark problem	√			√						√							
63	CEC2010_F14	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
64	CEC2010_F15	CEC'2010 constrained optimization benchmark problem	√			√						√							
65	CEC2010_F16	CEC'2010 constrained optimization benchmark problem	√			√						√							
66	CEC2010_F17	CEC'2010 constrained optimization benchmark problem	√			√						√							
67	CEC2010_F18	CEC'2010 constrained optimization benchmark problem	√			√						√							
68	CEC2013_F1	Shifted elliptic function	√			√					√								
69	CEC2013_F2	Shifted Rastrigin's function	√			√					√								
70	CEC2013_F3	Shifted Ackley's function	√			√					√								
71	CEC2013_F4	7-nonseparable, 1-separable shifted and rotated elliptic function	√			√					√								
72	CEC2013_F5	7-nonseparable, 1-separable shifted and rotated Rastrigin's function	√			√					√								
73	CEC2013_F6	7-nonseparable, 1-separable shifted and rotated Ackley's function	√			√					√								
74	CEC2013_F7	7-nonseparable, 1-separable shifted and rotated Schwefel's function	√			√					√								
75	CEC2013_F8	20-nonseparable shifted and rotated elliptic function	√			√					√								
76	CEC2013_F9	20-nonseparable shifted and rotated Rastrigin's function	√			√					√								
77	CEC2013_F10	20-nonseparable shifted and rotated Rastrigin's function	√			√					√								
78	CEC2013_F11	20-nonseparable shifted and rotated Schwefel's function	√			√					√								
79	CEC2013_F12	Shifted Rosenbrock's function	√			√					√								
80	CEC2013_F13	Shifted Schwefel's function with conforming overlapping subcomponents	√			√					√								
81	CEC2013_F14	Shifted Schwefel's function with conflicting overlapping subcomponents	√			√					√								
82	CEC2013_F15	Shifted Schwefel's function	√			√					√								
83	CEC2017_F1	CEC'2017 constrained optimization benchmark problem	√			√						√							
84	CEC2017_F2	CEC'2017 constrained optimization benchmark problem	√			√						√							
85	CEC2017_F3	CEC'2017 constrained optimization benchmark problem	√			√						√							
86	CEC2017_F4	CEC'2017 constrained optimization benchmark problem	√			√						√							
87	CEC2017_F5	CEC'2017 constrained optimization benchmark problem	√			√						√							
88	CEC2017_F6	CEC'2017 constrained optimization benchmark problem	√			√						√							
89	CEC2017_F7	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
90	CEC2017_F8	CEC'2017 constrained optimization benchmark problem	√			√						√							
91	CEC2017_F9	CEC'2017 constrained optimization benchmark problem	√			√						√							
92	CEC2017_F10	CEC'2017 constrained optimization benchmark problem	√			√						√							
93	CEC2017_F11	CEC'2017 constrained optimization benchmark problem	√			√						√							
94	CEC2017_F12	CEC'2017 constrained optimization benchmark problem	√			√						√							
95	CEC2017_F13	CEC'2017 constrained optimization benchmark problem	√			√						√							
96	CEC2017_F14	CEC'2017 constrained optimization benchmark problem	√			√						√							
97	CEC2017_F15	CEC'2017 constrained optimization benchmark problem	√			√						√							
98	CEC2017_F16	CEC'2017 constrained optimization benchmark problem	√			√						√							
99	CEC2017_F17	CEC'2017 constrained optimization benchmark problem	√			√						√							
100	CEC2017_F18	CEC'2017 constrained optimization benchmark problem	√			√						√							
101	CEC2017_F19	CEC'2017 constrained optimization benchmark problem	√			√						√							
102	CEC2017_F20	CEC'2017 constrained optimization benchmark problem	√			√						√							
103	CEC2017_F21	CEC'2017 constrained optimization benchmark problem	√			√						√							
104	CEC2017_F22	CEC'2017 constrained optimization benchmark problem	√			√						√							
105	CEC2017_F23	CEC'2017 constrained optimization benchmark problem	√			√						√							
106	CEC2017_F24	CEC'2017 constrained optimization benchmark problem	√			√						√							
107	CEC2017_F25	CEC'2017 constrained optimization benchmark problem	√			√						√							
108	CEC2017_F26	CEC'2017 constrained optimization benchmark problem	√			√						√							
109	CEC2017_F27	CEC'2017 constrained optimization benchmark problem	√			√						√							
110	CEC2017_F28	CEC'2017 constrained optimization benchmark problem	√			√						√							
111	CEC2020_F1	Bent cigar function	√			√													
112	CEC2020_F2	Shifted and rotated Schwefel's function	√			√													
113	CEC2020_F3	Shifted and rotated Lunacek bi-Rastrigin function	√			√													
114	CEC2020_F4	Expanded Rosenbrock's plus Griewangk's function	√			√													

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
115	CEC2020_F5	Hybrid function 1	√			√													
116	CEC2020_F6	Hybrid function 2	√			√													
117	CEC2020_F7	Hybrid function 3	√			√													
118	CEC2020_F8	Composition function 1	√			√													
119	CEC2020_F9	Composition function 2	√			√													
120	CEC2020_F10	Composition function 3	√			√													
121	CF1	Constrained benchmark MOP		√		√					√	√							
122	CF2	Constrained benchmark MOP		√		√					√	√							
123	CF3	Constrained benchmark MOP		√		√					√	√							
124	CF4	Constrained benchmark MOP		√		√					√	√							
125	CF5	Constrained benchmark MOP		√		√					√	√							
126	CF6	Constrained benchmark MOP		√		√					√	√							
127	CF7	Constrained benchmark MOP		√		√					√	√							
128	CF8	Constrained benchmark MOP		√		√					√	√							
129	CF9	Constrained benchmark MOP		√		√					√	√							
130	CF10	Constrained benchmark MOP		√		√					√	√							
131	CI_HS	Multitasking problem (Griewank function + Rastrigin function)	√			√					√						√		
132	CI_LS	Multitasking problem (Ackley function + Schwefel function)	√			√					√						√		
133	CI_MS	Multitasking problem (Ackley function + Rastrigin function)	√			√					√						√		
134	CitySegMOP1	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
135	CitySegMOP2	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
136	CitySegMOP3	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
137	CitySegMOP4	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
138	CitySegMOP5	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
139	CitySegMOP6	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
140	CitySegMOP7	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
141	CitySegMOP8	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
142	CitySegMOP9	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
143	CitySegMOP10	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
144	CitySegMOP11	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
145	CitySegMOP12	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
146	CitySegMOP13	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
147	CitySegMOP14	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
148	CitySegMOP15	Neural architecture search on Cityscape segmentation datasets		√		√					√		√						
149	Community Detection	The community detection problem with label based encoding	√					√			√		√						
150	DAS-CMOP1	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
151	DAS-CMOP2	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
152	DAS-CMOP3	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
153	DAS-CMOP4	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
154	DAS-CMOP5	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
155	DAS-CMOP6	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
156	DAS-CMOP7	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
157	DAS-CMOP8	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
158	DAS-CMOP9	Difficulty-adjustable and scalable constrained benchmark MOP		√		√					√	√							
159	DOC1	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
160	DOC2	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
161	DOC3	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
162	DOC4	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
163	DOC5	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
164	DOC6	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
165	DOC7	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
166	DOC8	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
167	DOC9	Benchmark MOP with constraints in decision and objective spaces		√		√						√							
168	DTLZ1	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
169	DTLZ2	Benchmark MOP proposed by Deb, Thiele,		√	√	√					√		√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		Laumanns, and Zitzler																	
170	DTLZ3	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
171	DTLZ4	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
172	DTLZ5	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
173	DTLZ6	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
174	DTLZ7	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
175	DTLZ8	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√	√	√						
176	DTLZ9	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√	√	√						
177	CDTLZ2	Convex DTLZ2		√	√	√					√		√						
178	IDTLZ1	Inverted DTLZ1		√	√	√					√		√						
179	IDTLZ2	Inverted DTLZ2		√	√	√					√		√						
180	SDTLZ1	Scaled DTLZ1		√	√	√					√		√						
181	SDTLZ2	Scaled DTLZ2		√	√	√					√		√						
182	C1-DTLZ1	Constrained DTLZ1		√	√	√					√	√	√						
183	C1-DTLZ3	Constrained DTLZ3		√	√	√					√	√	√						
184	C2-DTLZ2	Constrained DTLZ2		√	√	√					√	√	√						
185	C3-DTLZ4	Constrained DTLZ4		√	√	√					√	√	√						
186	DC1-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
187	DC1-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
188	DC2-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
189	DC2-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
190	DC3-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
191	DC3-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
192	FCP1	Benchmark constrained MOP proposed by Yuan		√		√						√							
193	FCP2	Benchmark constrained MOP proposed by Yuan		√		√						√							
194	FCP3	Benchmark constrained MOP proposed by Yuan		√		√						√							
195	FCP4	Benchmark constrained MOP proposed by Yuan		√		√						√							
196	FCP5	Benchmark constrained MOP proposed by Yuan		√		√						√							
197	FDA1	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
198	FDA2	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
199	FDA3	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
200	FDA4	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
201	FDA5	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
202	GLSMOP1	General large-scale benchmark MOP		√	√	√					√		√						
203	GLSMOP2	General large-scale benchmark MOP		√	√	√					√		√						
204	GLSMOP3	General large-scale benchmark MOP		√	√	√					√		√						
205	GLSMOP4	General large-scale benchmark MOP		√	√	√					√		√						
206	GLSMOP5	General large-scale benchmark MOP		√	√	√					√		√						
207	GLSMOP6	General large-scale benchmark MOP		√	√	√					√		√						
208	GLSMOP7	General large-scale benchmark MOP		√	√	√					√		√						
209	GLSMOP8	General large-scale benchmark MOP		√	√	√					√		√						
210	GLSMOP9	General large-scale benchmark MOP		√	√	√					√		√						
211	IMMOEA_F1	Benchmark MOP for testing IM-MOEA		√		√					√								
212	IMMOEA_F2	Benchmark MOP for testing IM-MOEA		√		√					√								
213	IMMOEA_F3	Benchmark MOP for testing IM-MOEA		√		√					√								
214	IMMOEA_F4	Benchmark MOP for testing IM-MOEA		√		√					√								
215	IMMOEA_F5	Benchmark MOP for testing IM-MOEA		√		√					√								
216	IMMOEA_F6	Benchmark MOP for testing IM-MOEA		√		√					√								
217	IMMOEA_F7	Benchmark MOP for testing IM-MOEA		√		√					√								
218	IMMOEA_F8	Benchmark MOP for testing IM-MOEA		√		√					√								
219	IMMOEA_F9	Benchmark MOP for testing IM-MOEA		√		√					√								
220	IMMOEA_F10	Benchmark MOP for testing IM-MOEA		√		√					√								
221	IMOP1	Benchmark MOP with irregular Pareto front		√		√							√						
222	IMOP2	Benchmark MOP with irregular Pareto front		√		√							√						
223	IMOP3	Benchmark MOP with irregular Pareto front		√		√							√						
224	IMOP4	Benchmark MOP with irregular Pareto front		√		√							√						
225	IMOP5	Benchmark MOP with irregular Pareto front		√		√							√						
226	IMOP6	Benchmark MOP with irregular Pareto front		√		√							√						
227	IMOP7	Benchmark MOP with irregular Pareto front		√		√							√						
228	IMOP8	Benchmark MOP with irregular Pareto front		√		√							√						
229	IN1KMOP1	Neural architecture search on ImageNet 1K		√		√					√		√						
230	IN1KMOP2	Neural architecture search on ImageNet 1K		√		√					√		√						
231	IN1KMOP3	Neural architecture search on ImageNet 1K		√		√					√		√						
232	IN1KMOP4	Neural architecture search on ImageNet 1K		√		√					√		√						
233	IN1KMOP5	Neural architecture search on ImageNet 1K		√		√					√		√						
234	IN1KMOP6	Neural architecture search on ImageNet 1K		√		√					√		√						
235	IN1KMOP7	Neural architecture search on ImageNet 1K		√		√					√		√						
236	IN1KMOP8	Neural architecture search on ImageNet 1K		√		√					√		√						
237	IN1KMOP9	Neural architecture search on ImageNet 1K		√		√					√		√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
238	Instance1	Multitasking multi-objective problem (ZDT4-R + ZDT4-G)		√		√					√						√		
239	Instance2	Multitasking multi-objective problem (ZDT4-RC + ZDT4-A)		√		√					√	√					√		
240	KP	The knapsack problem	√						√		√	√							
241	LIR-CMOP1	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
242	LIR-CMOP2	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
243	LIR-CMOP3	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
244	LIR-CMOP4	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
245	LIR-CMOP5	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
246	LIR-CMOP6	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
247	LIR-CMOP7	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
248	LIR-CMOP8	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
249	LIR-CMOP9	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
250	LIR-CMOP10	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
251	LIR-CMOP11	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
252	LIR-CMOP12	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
253	LIR-CMOP13	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
254	LIR-CMOP14	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
255	LRMOP1	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
256	LRMOP2	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
257	LRMOP3	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
258	LRMOP4	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
259	LRMOP5	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
260	LRMOP6	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
261	LSCM1	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
262	LSCM2	Large-scale constrained multiobjective benchmark problem		√		√					√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
263	LSCM3	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
264	LSCM4	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
265	LSCM5	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
266	LSCM6	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
267	LSCM7	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
268	LSCM8	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
269	LSCM9	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
270	LSCM10	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
271	LSCM11	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
272	LSCM12	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
273	LSMOP1	Large-scale benchmark MOP		√	√	√					√								
274	LSMOP2	Large-scale benchmark MOP		√	√	√					√								
275	LSMOP3	Large-scale benchmark MOP		√	√	√					√								
276	LSMOP4	Large-scale benchmark MOP		√	√	√					√								
277	LSMOP5	Large-scale benchmark MOP		√	√	√					√								
278	LSMOP6	Large-scale benchmark MOP		√	√	√					√								
279	LSMOP7	Large-scale benchmark MOP		√	√	√					√								
280	LSMOP8	Large-scale benchmark MOP		√	√	√					√								
281	LSMOP9	Large-scale benchmark MOP		√	√	√					√								
282	MaF1	Inverted DTLZ1		√	√	√					√								
283	MaF2	DTLZ2BZ		√	√	√					√								
284	MaF3	Convex DTLZ3		√	√	√					√								
285	MaF4	Inverted and scaled DTLZ3		√	√	√					√								
286	MaF5	Scaled DTLZ4		√	√	√					√								
287	MaF6	DTLZ5IM		√	√	√					√								
288	MaF7	DTLZ7		√	√	√					√								
289	MaF8	MP-DMP		√	√	√													
290	MaF9	ML-DMP		√	√	√													
291	MaF10	WFG1		√	√	√					√								
292	MaF11	WFG2		√	√	√					√								
293	MaF12	WFG9		√	√	√					√								
294	MaF13	P7		√	√	√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
295	MaF14	LSMOP3		√	√	√					√								
296	MaF15	Inverted LSMOP8		√	√	√					√								
297	MaOPP_binary	Many-objective pathfinding problem based on binary encoding			√				√		√		√						
298	MaOPP_real	Many-objective pathfinding problem based on real encoding			√	√					√		√						
299	Mario	Play with Mario	√				√	√											
300	MaxCut	The max-cut problem	√						√		√								
301	MLDMP	The multi-line distance minimization problem		√	√	√													
302	MMF1	Multi-modal multi-objective test function		√		√								√					
303	MMF2	Multi-modal multi-objective test function		√		√								√					
304	MMF3	Multi-modal multi-objective test function		√		√								√					
305	MMF4	Multi-modal multi-objective test function		√		√								√					
306	MMF5	Multi-modal multi-objective test function		√		√								√					
307	MMF6	Multi-modal multi-objective test function		√		√								√					
308	MMF7	Multi-modal multi-objective test function		√		√								√					
309	MMF8	Multi-modal multi-objective test function		√		√								√					
310	MMMOP1	Multi-modal multi-objective optimization problem		√	√	√								√					
311	MMMOP2	Multi-modal multi-objective optimization problem		√	√	√								√					
312	MMMOP3	Multi-modal multi-objective optimization problem		√	√	√								√					
313	MMMOP4	Multi-modal multi-objective optimization problem		√	√	√								√					
314	MMMOP5	Multi-modal multi-objective optimization problem		√	√	√								√					
315	MMMOP6	Multi-modal multi-objective optimization problem		√	√	√								√					
316	MMOP_HS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
317	MMOP_HS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
318	MMOP_LS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
319	MMOP_LS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
320	MMOP_MS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
321	MMOP_MS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
322	MMOP_NS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
323	MMOP_NS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
324	MOEADDE_F1	Benchmark MOP for testing MOEA/D-DE		√		√					√								
325	MOEADDE_F2	Benchmark MOP for testing MOEA/D-DE		√		√					√								
326	MOEADDE_F3	Benchmark MOP for testing MOEA/D-DE		√		√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
327	MOEADDE_F4	Benchmark MOP for testing MOEA/D-DE		√		√					√								
328	MOEADDE_F5	Benchmark MOP for testing MOEA/D-DE		√		√					√								
329	MOEADDE_F6	Benchmark MOP for testing MOEA/D-DE		√		√					√								
330	MOEADDE_F7	Benchmark MOP for testing MOEA/D-DE		√		√					√								
331	MOEADDE_F8	Benchmark MOP for testing MOEA/D-DE		√		√					√								
332	MOEADDE_F9	Benchmark MOP for testing MOEA/D-DE		√		√					√								
333	MOEADM2M_F1	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
334	MOEADM2M_F2	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
335	MOEADM2M_F3	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
336	MOEADM2M_F4	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
337	MOEADM2M_F5	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
338	MOEADM2M_F6	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
339	MOEADM2M_F7	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
340	MOKP	The multi-objective knapsack problem		√	√				√		√	√							
341	MONRP	The multi-objective next release problem		√					√		√								
342	MOTSP	The multi-objective traveling salesman problem		√	√					√	√								
343	MPDMP	The multi-point distance minimization problem		√	√	√													
344	mQAP	The multi-objective quadratic assignment problem		√	√					√	√								
345	MW1	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
346	MW2	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
347	MW3	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
348	MW4	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
349	MW5	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
350	MW6	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
351	MW7	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
352	MW8	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
353	MW9	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
354	MW10	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
355	MW11	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
356	MW12	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
357	MW13	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																	
358	MW14	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
359	NI_HS	Multitasking problem (Rosenbrock function + Rastrigin function)	√			√					√						√		
360	NI_MS	Multitasking problem (Griewank function + Weierstrass function)	√			√					√						√		
361	RMMEDA_F1	Benchmark MOP for testing RM-MEDA		√		√					√								
362	RMMEDA_F2	Benchmark MOP for testing RM-MEDA		√		√					√								
363	RMMEDA_F3	Benchmark MOP for testing RM-MEDA		√		√					√								
364	RMMEDA_F4	Benchmark MOP for testing RM-MEDA		√		√					√								
365	RMMEDA_F5	Benchmark MOP for testing RM-MEDA		√		√					√								
366	RMMEDA_F6	Benchmark MOP for testing RM-MEDA		√		√					√								
367	RMMEDA_F7	Benchmark MOP for testing RM-MEDA		√		√					√								
368	RMMEDA_F8	Benchmark MOP for testing RM-MEDA		√		√					√								
369	RMMEDA_F9	Benchmark MOP for testing RM-MEDA		√		√					√								
370	RMMEDA_F10	Benchmark MOP for testing RM-MEDA		√		√					√								
371	RWMOP1	Pressure vessal problem		√		√						√							
372	RWMOP2	Vibrating platform		√		√						√							
373	RWMOP3	Two bar truss design problem		√		√						√							
374	RWMOP4	Weldan beam design problem		√		√						√							
375	RWMOP5	Disc brake design problem		√		√						√							
376	RWMOP6	Speed reducer design problem		√		√						√							
377	RWMOP7	Gear train design problem		√		√						√							
378	RWMOP8	Car side impact design problem		√		√						√							
379	RWMOP9	Four bar plane truss		√		√						√							
380	RWMOP10	Two bar plane truss		√		√						√							
381	RWMOP11	Water resource management problem		√		√						√							
382	RWMOP12	Simply supported I-beam design		√		√						√							
383	RWMOP13	Gear box design		√		√						√							
384	RWMOP14	Multiple-disk clutch brake design problem		√		√						√							
385	RWMOP15	Spring design problem		√		√						√							
386	RWMOP16	Cantilever beam design problem		√		√						√							
387	RWMOP17	Bulk carriers design problem		√		√						√							
388	RWMOP18	Front rail design problem		√		√						√							
389	RWMOP19	Multi-product batch plant		√		√						√							
390	RWMOP20	Hydro-static thrust bearing design problem		√		√						√							
391	RWMOP21	Crash energy management for high-speed train		√		√						√							
392	RWMOP22	Haverly's pooling problem		√		√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
393	RWMOP23	Reactor network design		√		√						√							
394	RWMOP24	Heat exchanger network design		√		√						√							
395	RWMOP25	Process synthesis problem		√		√						√							
396	RWMOP26	Process sythesis and design problem		√		√						√							
397	RWMOP27	Process flow sheeting problem		√		√						√							
398	RWMOP28	Two reactor problem		√		√						√							
399	RWMOP29	Process synthesis problem		√		√						√							
400	RWMOP30	Synchronous pptimal pulse-width modulation of 3-level inverters		√		√						√							
401	RWMOP31	Synchronous pptimal pulse-width modulation of 5-level inverters		√		√						√							
402	RWMOP32	Synchronous pptimal pulse-width modulation of 7-level inverters		√		√						√							
403	RWMOP33	Synchronous pptimal pulse-width modulation of 9-level inverters		√		√						√							
404	RWMOP34	Synchronous pptimal pulse-width modulation of 11-level inverters		√		√						√							
405	RWMOP35	Synchronous pptimal pulse-width modulation of 13-level inverters		√		√						√							
406	RWMOP36	Optimal sizing of single phase distributed generation with reactive power support for phase balancing at main transformer/grid and active power loss		√		√						√							
407	RWMOP37	Optimal Sizing of Single Phase Distributed Generation with reactive power support for Phase Balancing at Main Transformer/Grid and reactive Power loss		√		√						√							
408	RWMOP38	Optimal sizing of single phase distributed generation with reactive power support for active and reactive power loss		√		√						√							
409	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		√		√						√							
410	RWMOP40	Optimal power flow for minimizing active and reactive power loss		√		√						√							
411	RWMOP41	Optimal power flow for minimizing voltage deviation, active and reactive power loss		√		√						√							
412	RWMOP42	Optimal power flow for minimizing voltage deviation, and active power loss		√		√						√							
413	RWMOP43	Optimal power flow for minimizing fuel cost, and active power loss		√		√						√							
414	RWMOP44	Optimal power flow for minimizing fuel cost, active and reactive power loss		√		√						√							
415	RWMOP45	Optimal power flow for minimizing fuel cost, voltage deviation, and active power loss		√		√						√							
416	RWMOP46	Optimal power flow for minimizing fuel cost, voltage deviation, active and reactive power loss		√		√						√							
417	RWMOP47	Optimal droop setting for minimizing active		√		√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		and reactive power loss																	
418	RWMOP48	Optimal droop setting for minimizing voltage deviation and active power loss		√		√						√							
419	RWMOP49	Optimal droop setting for minimizing voltage deviation, active, and reactive power loss		√		√						√							
420	RWMOP50	Power distribution system planning		√		√						√							
421	SDC1	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
422	SDC2	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
423	SDC3	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
424	SDC4	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
425	SDC5	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
426	SDC6	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
427	SDC7	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
428	SDC8	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
429	SDC9	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
430	SDC10	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
431	SDC11	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
432	SDC12	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
433	SDC13	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
434	SDC14	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
435	SDC15	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
436	SMD1	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
437	SMD2	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
438	SMD3	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
439	SMD4	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
440	SMD5	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
441	SMD6	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
442	SMD7	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
443	SMD8	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
444	SMD9	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
445	SMD10	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
446	SMD11	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
447	SMD12	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
448	SO_ISCSO_2016	International student competition in structural optimization	√				√				√	√							
449	SO_ISCSO_2017	International student competition in structural optimization	√				√				√	√							
450	SO_ISCSO_2018	International student competition in structural optimization	√				√				√	√							
451	SO_ISCSO_2019	International student competition in structural optimization	√				√				√	√							
452	SO_ISCSO_2021	International student competition in structural optimization	√				√				√	√							
453	SO_ISCSO_2022	International student competition in structural optimization	√				√				√	√							
454	Sparse_CD	The community detection problem		√					√		√		√		√				
455	Sparse_CN	The critical node detection problem		√					√		√		√		√				
456	Sparse_FS	The feature selection problem		√					√		√		√		√				
457	Sparse_IS	The instance selection problem		√					√		√		√		√				
458	Sparse_KP	The sparse multi-objective knapsack problem		√	√				√		√								
459	Sparse_NN	The neural network training problem		√		√					√		√		√				
460	Sparse_PM	The pattern mining problem		√					√		√		√		√				
461	Sparse_PO	The portfolio optimization problem		√		√					√		√		√				
462	Sparse_SR	The sparse signal reconstruction problem		√		√					√		√		√				
463	SMMOP1	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
464	SMMOP2	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
465	SMMOP3	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
466	SMMOP4	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
467	SMMOP5	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
468	SMMOP6	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
469	SMMOP7	Sparse multi-modal multi-objective		√	√	√					√			√	√				

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		optimization problem																	
470	SMMOP8	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
471	SMOP1	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
472	SMOP2	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
473	SMOP3	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
474	SMOP4	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
475	SMOP5	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
476	SMOP6	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
477	SMOP7	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
478	SMOP8	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
479	SOP_F1	Sphere function	√			√							√						
480	SOP_F2	Schwefel's function 2.22	√			√							√						
481	SOP_F3	Schwefel's function 1.2	√			√							√						
482	SOP_F4	Schwefel's function 2.21	√			√							√						
483	SOP_F5	Generalized Rosenbrock's function	√			√							√						
484	SOP_F6	Step function	√			√							√						
485	SOP_F7	Quartic function with noise	√			√							√						
486	SOP_F8	Generalized Schwefel's function 2.26	√			√							√						
487	SOP_F9	Generalized Rastrigin's function	√			√							√						
488	SOP_F10	Ackley's function	√			√							√						
489	SOP_F11	Generalized Griewank's function	√			√							√						
490	SOP_F12	Generalized penalized function	√			√							√						
491	SOP_F13	Generalized penalized function	√			√							√						
492	SOP_F14	Shekel's foxholes function	√			√							√						
493	SOP_F15	Kowalik's function	√			√							√						
494	SOP_F16	Six-hump camel-back function	√			√							√						
495	SOP_F17	Branin function	√			√							√						
496	SOP_F18	Goldstein-price function	√			√							√						
497	SOP_F19	Hartman's family	√			√							√						
498	SOP_F20	Hartman's family	√			√							√						
499	SOP_F21	Shekel's family	√			√							√						
500	SOP_F22	Shekel's family	√			√							√						
501	SOP_F23	Shekel's family	√			√							√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
502	TP1	Test problem for robust multi-objective optimization		√		√					√								√
503	TP2	Test problem for robust multi-objective optimization		√		√					√								√
504	TP3	Test problem for robust multi-objective optimization		√		√					√								√
505	TP4	Test problem for robust multi-objective optimization		√		√					√								√
506	TP5	Test problem for robust multi-objective optimization		√		√					√								√
507	TP6	Test problem for robust multi-objective optimization		√		√					√								√
508	TP7	Test problem for robust multi-objective optimization		√		√					√								√
509	TP8	Test problem for robust multi-objective optimization		√		√					√								√
510	TP9	Test problem for robust multi-objective optimization		√		√					√								√
511	TP10	Test problem for robust multi-objective optimization		√		√					√	√							√
512	TREE1	The time-varying ratio error estimation problem		√		√					√	√	√						
513	TREE2	The time-varying ratio error estimation problem		√		√					√	√	√						
514	TREE3	The time-varying ratio error estimation problem		√		√					√	√	√						
515	TREE4	The time-varying ratio error estimation problem		√		√					√	√	√						
516	TREE5	The time-varying ratio error estimation problem		√		√					√	√	√						
517	TREE6	The time-varying ratio error estimation problem		√		√					√	√	√						
518	TSP	The traveling salesman problem	√							√	√								
519	UF1	Unconstrained benchmark MOP		√		√					√								
520	UF2	Unconstrained benchmark MOP		√		√					√								
521	UF3	Unconstrained benchmark MOP		√		√					√								
522	UF4	Unconstrained benchmark MOP		√		√					√								
523	UF5	Unconstrained benchmark MOP		√		√					√								
524	UF6	Unconstrained benchmark MOP		√		√					√								
525	UF7	Unconstrained benchmark MOP		√		√					√								
526	UF8	Unconstrained benchmark MOP		√		√					√								
527	UF9	Unconstrained benchmark MOP		√		√					√								
528	UF10	Unconstrained benchmark MOP		√		√					√								
529	VNT1	Benchmark MOP proposed by Viennet		√		√													
530	VNT2	Benchmark MOP proposed by Viennet		√		√													
531	VNT3	Benchmark MOP proposed by Viennet		√		√													
532	VNT4	Benchmark MOP proposed by Viennet		√		√						√							
533	WFG1	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
534	WFG2	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
535	WFG3	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
536	WFG4	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
537	WFG5	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
538	WFG6	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
539	WFG7	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
540	WFG8	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
541	WFG9	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
542	ZCAT1	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
543	ZCAT2	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
544	ZCA3	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
545	ZCA4	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
546	ZCA5	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
547	ZCAT6	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
548	ZCAT7	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
549	ZCAT8	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
550	ZCAT9	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
551	ZCAT10	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
552	ZCAT11	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
553	ZCAT12	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
554	ZCAT13	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
555	ZCAT14	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
556	ZCAT15	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
557	ZCAT16	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
558	ZCAT17	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
559	ZCAT18	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
560	ZCAT19	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
561	ZCAT20	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
562	ZDT1	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
563	ZDT2	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
564	ZDT3	Benchmark MOP proposed by Zitzler, Deb,		√		√					√		√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		and Thiele																	
565	ZDT4	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
566	ZDT5	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√					√		√		√						
567	ZDT6	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
568	ZXH_CF1	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
569	ZXH_CF2	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
570	ZXH_CF3	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
571	ZXH_CF4	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
572	ZXH_CF5	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
573	ZXH_CF6	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
574	ZXH_CF7	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
575	ZXH_CF8	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
576	ZXH_CF9	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
577	ZXH_CF10	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
578	ZXH_CF11	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
579	ZXH_CF12	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
580	ZXH_CF13	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
581	ZXH_CF14	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
582	ZXH_CF15	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
583	ZXH_CF16	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							