PlatEMO

Evolutionary Multi-Objective Optimization Platform

User Manual 4.13

BIMK Group

July 20, 2025

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

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

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

If you have any comment or suggestion to PlatEMO, please send it to *field910921@gmail.com* (*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

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

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 $l_i \leq x_i \leq u_i$. $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_M(\mathbf{x})$ 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:

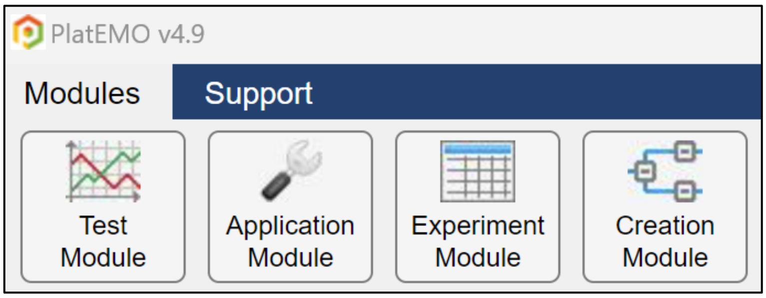
```
f1 = @(x)sum(x);
f2 = @(x)1-sum(x);
platemo('objFcn',f1,'conFcn',f2,'algorithm',@GA);
```

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.

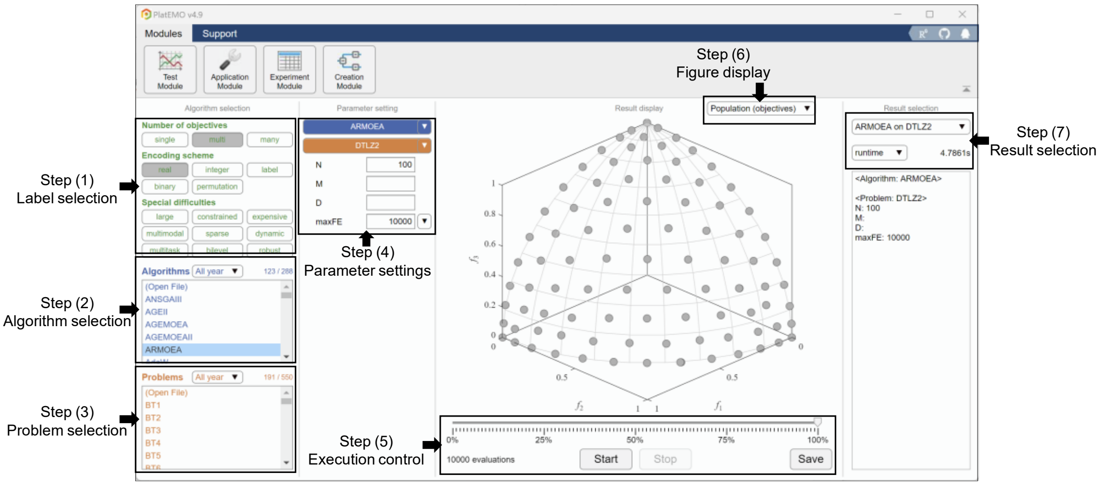
III. Using PlatEMO with GUI

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.

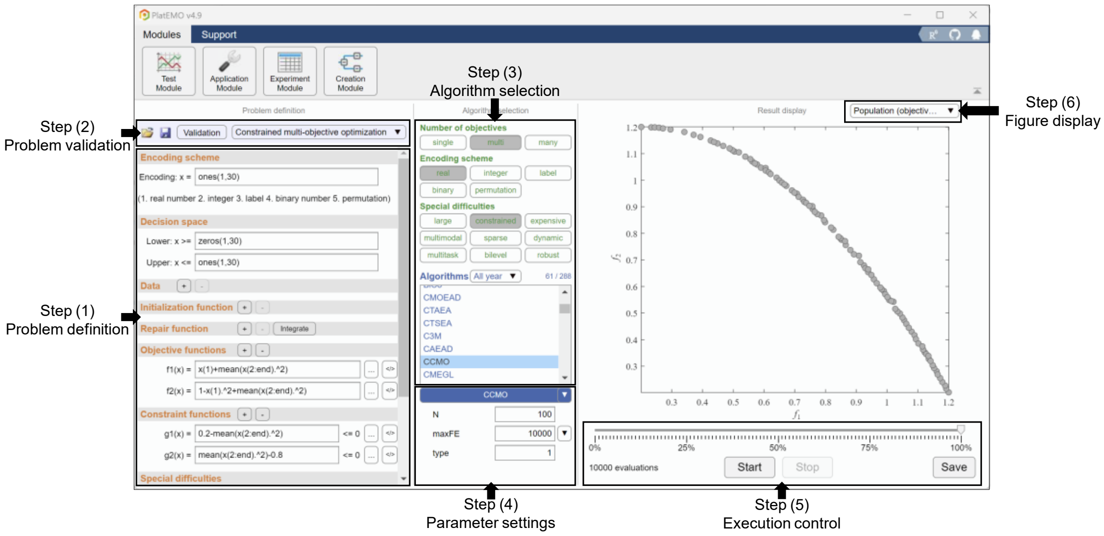


In this module, the performance investigation can be achieved by the following steps:

- 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.

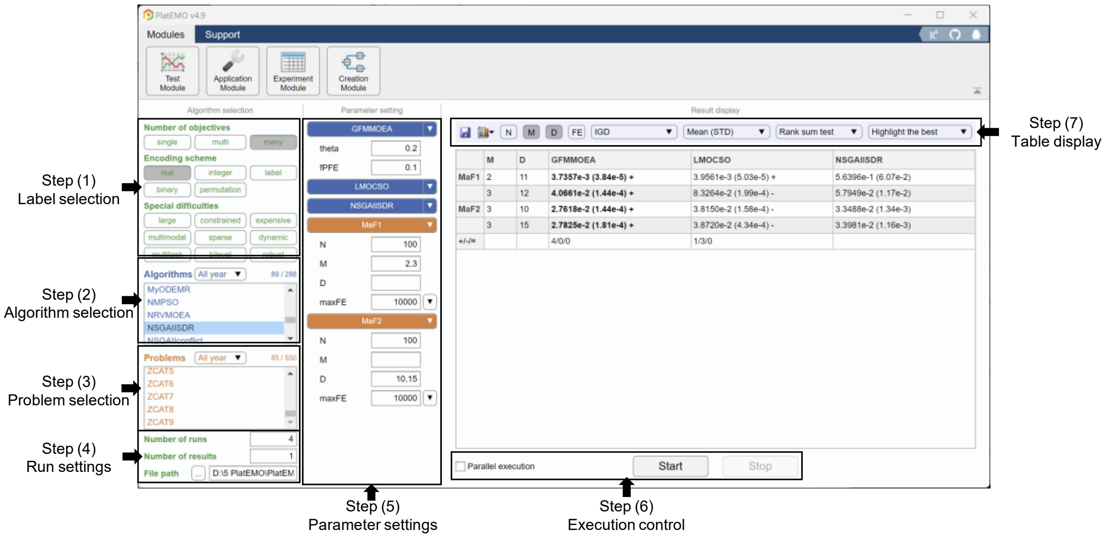


In this module, the solving of problems can be achieved by the following steps:

- 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.

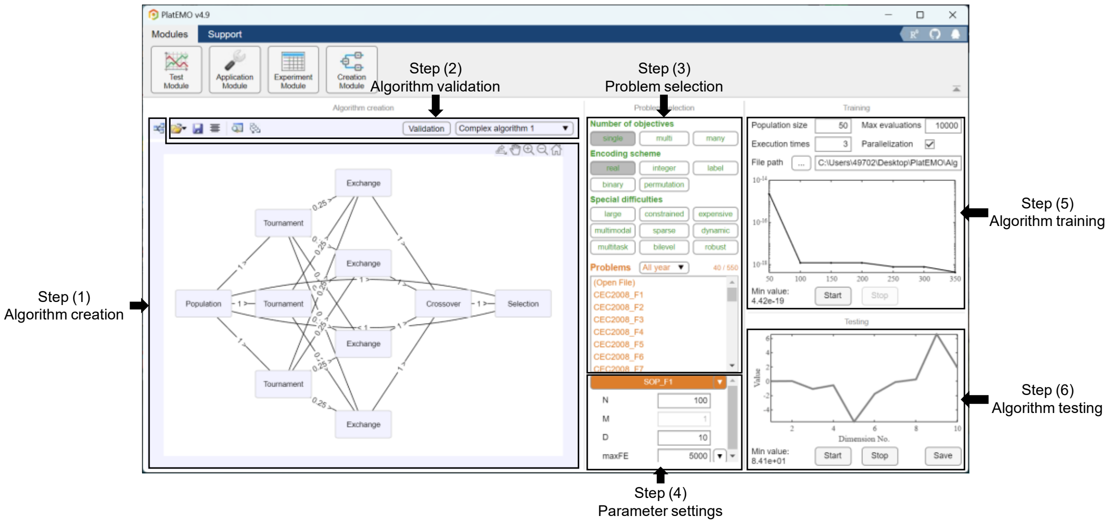


In this module, comparative experiments can be achieved by the following steps:

- 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
`RouletteWheelSelection`	Roulette-wheel selection
`TournamentSelection`	Tournament selection
`UniformPoint`	Generate a set of uniformly distributed points

B. PROBLEM Class

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

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

		population. A constraint is satisfied if and only if the constraint violation is not positive Input: A matrix consisting of decision vectors Output: A matrix consisting of constraint violations
`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	AE-NSGA-II	Autoencoding NSGA-II		√		√	√	√	√	√		√				√			
9	AESSPSO	Adaptive exploration state-space particle swarm optimization	√			√	√				√	√							
10	AFSEA	Adjoint feature-selection-based evolutionary algorithm		√		√	√		√		√	√			√				
11	AGE-II	Approximation-guided evolutionary multi-objective algorithm II		√		√	√	√	√	√									
12	AGE-MOEA	Adaptive geometry estimation-based many-objective evolutionary algorithm		√	√	√	√	√	√	√		√							
13	AGE-MOEA-II	Adaptive geometry estimation-based many-objective evolutionary algorithm II		√	√	√	√	√	√	√		√							
14	AGSEA	Automated guiding vector selection-based evolutionary algorithm		√		√	√		√		√	√			√				
15	A-NSGA-III	Adaptive NSGA-III		√	√	√	√	√	√	√		√							
16	APSEA	Adaptive population sizing based evolutionary algorithm		√		√	√	√	√	√		√							
17	AR-MOEA	Adaptive reference points based multi-objective evolutionary algorithm		√	√	√	√	√	√	√		√							
18	AutoV	Automated design of variation operators	√	√		√	√				√	√							
19	AVG-SAEA	Adaptive variable grouping based surrogate-assisted evolutionary algorithm		√		√	√				√		√						
20	BCE-IBEA	Bi-criterion evolution based IBEA		√	√	√	√	√	√	√									
21	BCE-MOEA/D	Bi-criterion evolution based MOEA/D		√	√	√	√	√	√	√									
22	BFGS	A quasi-Newton method proposed by Broyden, Fletcher, Goldfarb, and Shanno	√			√					√								
23	BiCo	Bidirectional coevolution constrained multiobjective evolutionary algorithm		√		√	√	√	√	√		√							
24	BiGE	Bi-goal evolution			√	√	√	√	√	√									
25	BLEAQII	Bilevel evolutionary algorithm based on		√		√						√						√	

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

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

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

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		objective evolutionary algorithm																	
109	GDE3	Generalized differential evolution 3		√		√	√					√							
110	GFM-MOEA	Generic front modeling based multi-objective evolutionary algorithm		√	√	√	√	√	√	√									
111	GLMO	Grouped and linked mutation operator algorithm		√		√	√				√								
112	g-NSGA-II	g-dominance based NSGA-II		√		√	√	√	√	√									
113	GPSO	Gradient based particle swarm optimization algorithm	√			√					√	√							
114	GPSOM	Gradient based particle swarm optimization algorithm (for multi-objective optimization)		√	√	√					√	√							
115	GrEA	Grid-based evolutionary algorithm			√	√	√	√	√	√									
116	GWASF-GA	Global weighting achievement scalarizing function genetic algorithm		√		√	√	√	√	√									
117	GWO	Grey wolf optimizer	√			√	√				√	√							
118	HEA	Hyper-dominance based evolutionary algorithm		√	√	√			√	√									
119	HeE-MOEA	Multiobjective evolutionary algorithm with heterogeneous ensemble based infill criterion		√		√	√						√						
120	HHC-MMEA	Hybrid hierarchical clustering based multi-modal multi-objective evolutionary algorithm		√		√					√			√	√				
121	hpaEA	Hyperplane assisted evolutionary algorithm		√	√	√	√	√	√	√									
122	HREA	Hierarchy ranking based evolutionary algorithm		√		√	√							√					
123	HypE	Hypervolume estimation algorithm		√	√	√	√	√	√	√									
124	IBEA	Indicator-based evolutionary algorithm		√	√	√	√	√	√	√									
125	ICMA	Indicator based constrained multi-objective algorithm		√		√	√					√							
126	I-DBEA	Improved decomposition-based evolutionary algorithm		√	√	√	√	√	√	√		√							
127	IM-C-MOEA/D	Inverse modeling constrained MOEA/D		√		√	√				√	√							
128	IM-MOEA	Inverse modeling based multiobjective evolutionary algorithm		√		√	√				√								
129	IM-MOEA/D	Inverse modeling MOEA/D		√		√	√				√								
130	IMODE	Improved multi-operator differential evolution	√			√	√				√	√							
131	IMTCMO	Improved evolutionary multitasking-based CMOEA		√		√	√	√	√	√		√							
132	IMTCMO_BS	Improved evolutionary multitasking-based CMOEA with bidirectional sampling		√	√	√	√	√	√	√		√							
133	I-SIBEA	Interactive simple indicator-based evolutionary algorithm		√		√	√	√	√	√									
134	Izui	An aggregative gradient based multi-objective optimizer proposed by Izui et al.		√	√	√					√	√							
135	KLEA	Knowledge learning-based evolutionary algorithm		√		√	√		√		√	√			√				
136	KL-NSGA-II	Knowledge learning based NSGA-II		√		√	√	√	√	√		√				√			
137	KMA	Komodo mlipir algorithm	√			√	√				√	√							
138	KnEA	Knee point driven evolutionary algorithm			√	√	√	√	√	√		√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
139	K-RVEA	Surrogate-assisted RVEA		√	√	√	√						√						
140	KTA2	Kriging-assisted Two_Arch2		√	√	√	√						√						
141	KTS	Kriging-assisted evolutionary algorithm with two search modes		√	√		√					√	√						
142	L2SMEA	Linear subspace surrogate modeling assisted evolutionary algorithm	√			√							√						
143	LCMEA	Large-scale constrained multi-objective evolutionary algorithm		√		√					√	√							
144	LCSA	Linear combination-based search algorithm		√	√	√	√				√								
145	LDS-AF	Low-dimensional surrogate aggregation function		√		√	√				√		√						
146	LERD	Large-scale evolutionary algorithm with reformulated decision variable analysis		√	√	√					√								
147	LMEA	Evolutionary algorithm for large-scale many-objective optimization		√	√	√	√				√								
148	LMOCSO	Large-scale multi-objective competitive swarm optimization algorithm		√	√	√	√				√	√							
149	LMOEA-DS	Large-scale evolutionary multi-objective optimization assisted by directed sampling		√		√	√				√								
150	LMPFE	Evolutionary algorithm with local model based Pareto front estimation		√	√	√	√	√	√	√									
151	LRMOEA	Large-scale robust multi-objective evolutionary algorithm		√		√			√		√	√			√				√
152	LSMOF	Large-scale multi-objective optimization framework with NSGA-II		√		√	√				√								
153	MaOEA-CSS	Many-objective evolutionary algorithms based on coordinated selection		√	√	√	√	√	√	√									
154	MaOEA-DDFC	Many-objective evolutionary algorithm based on directional diversity and favorable convergence		√	√	√	√	√	√	√									
155	MaOEA/IGD	IGD based many-objective evolutionary algorithm			√	√	√	√	√	√									
156	MaOEA/IT	Many-objective evolutionary algorithms based on an independent two-stage		√	√	√	√					√							
157	MaOEA-R&D	Many-objective evolutionary algorithm based on objective space reduction			√	√	√	√	√	√									
158	MCCMO	Multi-population coevolutionary constrained multi-objective optimization		√		√	√	√	√	√		√							
159	MCEA/D	Multiple classifiers-assisted evolutionary algorithm based on decomposition		√	√	√	√						√						
160	MFEA	Multifactorial evolutionary algorithm	√			√	√	√	√	√	√						√		
161	MFEA-II	Multifactorial evolutionary algorithm II	√			√	√	√	√	√	√						√		
162	MFFS	Multiform feature selection		√					√										
163	MFO-SPEA2	Multiform optimization framework based on SPEA2		√		√	√	√	√	√		√							
164	MGCEA	Multi-granularity clustering based evolutionary algorithm		√		√			√		√	√			√				
165	MGO	Mountain gazelle optimizer	√			√	√				√	√							
166	MGSAEA	Multigranularity surrogate-assisted constrained evolutionary algorithm		√		√						√	√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
167	MMEAPSL	Multimodal multi-objective evolutionary algorithm assisted by Pareto set learning		√		√	√	√	√	√				√					
168	MMEA-WI	Weighted indicator-based evolutionary algorithm for multimodal multi-objective optimization		√		√	√							√					
169	MMOPSO	MOPSO with multiple search strategies		√		√	√												
170	MO_Ring_ PSO_SCD	Multiobjective PSO using ring topology and special crowding distance		√		√	√							√					
171	MOBCA	Multi-objective besiege and conquer algorithm		√		√	√												
172	MOCell	Cellular genetic algorithm		√		√	√	√	√	√		√							
173	MOCGDE	Multi-objective conjugate gradient and differential evolution algorithm		√	√	√					√	√							
174	MO-CMA	Multi-objective covariance matrix adaptation evolution strategy		√		√	√												
175	MOEA/CKF	Multi-objective evolutionary algorithm based on cross-scale knowledge fusion		√		√			√		√	√			√				
176	MOEA/D	Multiobjective evolutionary algorithm based on decomposition		√	√	√	√	√	√	√									
177	MOEA/D-2WA	MOEA/D with two-type weight vector adjustments		√	√	√	√	√	√	√		√							
178	MOEA/D-AWA	MOEA/D with adaptive weight adjustment		√	√	√	√	√	√	√									
179	MOEA/D-CMA	MOEA/D with covariance matrix adaptation evolution strategy		√	√	√	√												
180	MOEA/D-CMT	MOEA/D with competitive multitasking		√		√						√							
181	MOEA/DD	Many-objective evolutionary algorithm based on dominance and decomposition		√	√	√	√	√	√	√		√							
182	MOEA/D-DAE	MOEA/D with detect-and-escape strategy		√		√	√	√	√	√		√							
183	MOEA/D-DCWV	MOEA/D with distribution control of weight vector set		√	√	√	√	√	√	√									
184	MOEA/D-DE	MOEA/D based on differential evolution		√	√	√	√												
185	MOEA/D-DQN	MOEA/D based on deep Q-network		√	√	√	√												
186	MOEA/D-DRA	MOEA/D with dynamical resource allocation		√	√	√	√												
187	MOEA/D-DU	MOEA/D with a distance based updating strategy		√	√	√	√	√	√	√									
188	MOEA/D-DYTS	MOEA/D with dynamic Thompson sampling		√	√	√	√												
189	MOEA/D-EGO	MOEA/D with efficient global optimization		√		√	√						√						
190	MOEA/D-FRRMAB	MOEA/D with fitness-rate-rank-based multiarmed bandit		√	√	√	√												
191	MOEA/D-M2M	MOEA/D based on MOP to MOP		√		√	√												
192	MOEA/D-MRDL	MOEA/D with maximum relative diversity loss		√		√	√												
193	MOEA/D-PaS	MOEA/D with Pareto adaptive scalarizing approximation		√	√	√	√												
194	MOEA/D-PFE	MOEA/D with Pareto front estimation		√	√	√	√	√	√	√									
195	MOEA/D-STM	MOEA/D with stable matching		√	√	√	√												
196	MOEA/D-UR	MOEA/D with update when required		√	√	√	√	√	√	√									

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

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

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
254	PEAplus	Pareto-based Kriging-assisted constrained multiobjective evolutionary algorithm plus		√		√	√					√	√						
255	PeEA	Pareto front shape estimation based evolutionary algorithm		√	√	√	√	√	√	√									
256	PESA-II	Pareto envelope-based selection algorithm II		√		√	√	√	√	√									
257	PICEA-g	Preference-inspired coevolutionary algorithm with goals		√	√	√	√	√	√	√									
258	PIEA	Performance indicator-based evolutionary algorithm		√	√	√							√						
259	PIMD	Probability and mapping crowding distance		√	√	√							√						
260	PM-MOEA	Pattern mining based multi-objective evolutionary algorithm		√		√	√		√		√	√			√				
261	POCEA	Paired offspring generation based constrained evolutionary algorithm		√		√	√				√	√							
262	PPS	Push and pull search algorithm		√	√	√	√					√							
263	PRDH	Problem reformulation and duplication handling		√					√										
264	PREA	Promising-region based EMO algorithm		√	√	√	√	√	√	√									
265	PSO	Particle swarm optimization	√			√	√				√	√							
266	REMO	Expensive multiobjective optimization by relation learning and prediction		√	√	√							√						
267	RGA-M1-2	Real-coded genetic algorithm with framework M1-2		√		√						√	√						
268	RGA-M2-2	Real-coded genetic algorithm with framework M2-2		√		√						√	√						
269	RM-MEDA	Regularity model-based multiobjective estimation of distribution		√		√	√												
270	RMOEA/DVA	Robust multi-objective evolutionary algorithm with decision variable assortment		√		√	√												√
271	RMSProp	Root mean square propagation	√			√					√								
272	r-NSGA-II	r-dominance based NSGA-II		√		√	√	√	√	√									
273	RPD-NSGA-II	Reference point dominance-based NSGA-II		√	√	√	√	√	√	√									
274	RPEA	Reference points-based evolutionary algorithm			√	√	√	√	√	√									
275	RSEA	Radial space division based evolutionary algorithm		√	√	√	√	√	√	√									
276	RVEA	Reference vector guided evolutionary algorithm		√	√	√	√	√	√	√		√							
277	RVEAa	RVEA embedded with the reference vector regeneration strategy			√	√	√	√	√	√									
278	RVEA-iGNG	RVEA based on improved growing neural gas		√	√	√	√	√	√	√									
279	S3-CMA-ES	Scalable small subpopulations based covariance matrix adaptation		√	√	√	√				√								
280	SA	Simulated annealing	√			√	√				√	√							
281	SACC-EAM-II	Surrogate-assisted cooperative co-evolutionary algorithm of Minamo	√			√	√						√						
282	SACOSO	Surrogate-assisted cooperative swarm optimization	√			√	√				√		√						
283	SADE-AMSS	Surrogate-assisted differential evolution with adaptive multi-subspace search	√			√	√						√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
284	SADE-ATDSC	Surrogate-assisted differential evolution with adaptation of training data selection criterion	√			√	√						√						
285	SADE-Sammon	Sammon mapping assisted differential evolution	√			√	√						√						
286	SAMSO	Multiswarm-assisted expensive optimization	√			√	√				√		√						
287	SAPO	Surrogate-assisted partial optimization	√			√	√					√	√						
288	S-CDAS	Self-controlling dominance area of solutions			√	√	√	√	√	√									
289	SCEA	Sparsity clustering basec evolutionary algorithm		√		√			√		√	√			√				
290	SD	Steepest descent	√			√					√								
291	S-ECSO	Enhanced competitive swarm optimizer for sparse optimization		√		√					√				√				
292	SFADE	Scalarization function approximation based differential evolution algorithm		√	√	√	√						√						
293	SGEA	Steady-state and generational evolutionary algorithm		√		√	√	√	√	√		√				√			
294	SGECF	Sparsity-guided elitism co-evolutionary framework		√		√			√		√	√			√				
295	SHADE	Success-history based adaptive differential evolution	√			√	√				√	√							
296	SIBEA	Simple indicator-based evolutionary algorithm		√		√	√	√	√	√									
297	SIBEA-kEMOSS	SIBEA with minimum objective subset of size k with minimum error			√	√	√	√	√	√									
298	SLMEA	Super-large-scale multi-objective evolutionary algorithm		√		√	√		√		√	√			√				
299	SMEA	Self-organizing multiobjective evolutionary algorithm		√		√	√												
300	SMOA	Supervised multi-objective optimization algorithm		√		√							√						
301	SMPSO	Speed-constrained multi-objective particle swarm optimization		√		√	√												
302	SMS-EGO	S metric selection based efficient global optimization		√		√	√						√						
303	SMS-EMOA	S metric selection based evolutionary multiobjective optimization		√		√	√	√	√	√									
304	S-NSGA-II	Sparse NSGA-II		√		√					√	√			√				
305	SparseEA	Evolutionary algorithm for sparse multi-objective optimization problems		√		√	√		√		√	√			√				
306	SparseEA2	Improved SparseEA		√		√	√		√		√	√			√				
307	SPEA2	Strength Pareto evolutionary algorithm 2		√		√	√	√	√	√									
308	SPEA2+SDE	SPEA2 with shift-based density estimation			√	√	√	√	√	√									
309	SPEA/R	Strength Pareto evolutionary algorithm based on reference direction		√	√	√	√	√	√	√									
310	SQP	Sequential quadratic programming	√			√					√	√							
311	SRA	Stochastic ranking algorithm			√	√	√	√	√	√									
312	SSCEA	Subspace segmentation based co-evolutionary algorithm		√	√	√	√												
313	SSDE	Self-organized surrogate-assisted differential evolution		√	√	√	√					√	√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
314	SSIO-RL	Search space independent operator based deep reinforcement learning	√			√	√				√	√							
315	SVR-NSGA-II	Support vector regression based NSGA-II		√		√	√	√	√	√		√				√			
316	t-DEA	theta-dominance based evolutionary algorithm		√	√	√	√	√	√	√									
317	tDEA-CPBI	Theta-dominance based evolutionary algorithm with CPBI		√	√	√	√	√	√	√		√							
318	TEA	Two-phase evolutionary algorithm		√	√	√	√					√	√						
319	TELSO	Two-layer encoding learning swarm optimizer		√		√			√		√	√			√				
320	TiGE-2	Tri-Goal Evolution Framework for CMaOPs			√	√	√	√	√	√		√							
321	ToP	Two-phase framework with NSGA-II		√		√	√					√							
322	TPCMaO	Three-population based constrained many-objective co-evolutionary algorithm			√	√	√	√	√	√		√							
323	TriMOEA-TA&R	Multi-modal MOEA using two-archive and recombination strategies		√		√	√							√					
324	TS-NSGA-II	Two-stage NSGA-II		√	√	√	√	√	√	√									
325	TS-SparseEA	Two-stage SparseEA		√		√			√		√	√			√				
326	TSTI	Two-stage evolutionary algorithm with three indicators		√		√	√	√	√	√		√							
327	Two_Arch2	Two-archive algorithm 2		√	√	√	√	√	√	√									
328	URCMO	Utilizing the relationship between constrained and unconstrained Pareto fronts for constrained multi-objective optimization		√		√	√					√							
329	VaEA	Vector angle based evolutionary algorithm		√	√	√	√	√	√	√									
330	WASF-GA	Weighting achievement scalarizing function genetic algorithm		√		√	√	√	√	√									
331	WOA	Whale optimization algorithm	√			√	√				√	√							
332	WOF	Weighted optimization framework		√		√	√				√								
333	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	DSMOP1	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
169	DSMOP2	Dynamic sparse multi-objective optimization		√	√	√					√				√	√			

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		problem																	
170	DSMOP3	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
171	DSMOP4	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
172	DSMOP5	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
173	DSMOP6	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
174	DSMOP7	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
175	DSMOP8	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
176	DSMOP9	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
177	DSMOP10	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
178	DSMOP11	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
179	DSMOP12	Dynamic sparse multi-objective optimization problem		√	√	√					√				√	√			
180	DTLZ1	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
181	DTLZ2	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
182	DTLZ3	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
183	DTLZ4	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
184	DTLZ5	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
185	DTLZ6	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
186	DTLZ7	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√		√						
187	DTLZ8	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√	√	√						
188	DTLZ9	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√					√	√	√						
189	CDTLZ2	Convex DTLZ2		√	√	√					√		√						
190	IDTLZ1	Inverted DTLZ1		√	√	√					√		√						
191	IDTLZ2	Inverted DTLZ2		√	√	√					√		√						
192	SDTLZ1	Scaled DTLZ1		√	√	√					√		√						
193	SDTLZ2	Scaled DTLZ2		√	√	√					√		√						
194	C1-DTLZ1	Constrained DTLZ1		√	√	√					√	√	√						
195	C1-DTLZ3	Constrained DTLZ3		√	√	√					√	√	√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
196	C2-DTLZ2	Constrained DTLZ2		√	√	√					√	√	√						
197	C3-DTLZ4	Constrained DTLZ4		√	√	√					√	√	√						
198	DC1-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
199	DC1-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
200	DC2-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
201	DC2-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
202	DC3-DTLZ1	DTLZ1 with constrains in decision space		√	√	√					√	√	√						
203	DC3-DTLZ3	DTLZ3 with constrains in decision space		√	√	√					√	√	√						
204	FCP1	Benchmark constrained MOP proposed by Yuan		√		√						√							
205	FCP2	Benchmark constrained MOP proposed by Yuan		√		√						√							
206	FCP3	Benchmark constrained MOP proposed by Yuan		√		√						√							
207	FCP4	Benchmark constrained MOP proposed by Yuan		√		√						√							
208	FCP5	Benchmark constrained MOP proposed by Yuan		√		√						√							
209	FDA1	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
210	FDA2	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
211	FDA3	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
212	FDA4	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
213	FDA5	Benchmark dynamic MOP proposed by Farina, Deb, and Amato		√		√					√					√			
214	GLSMOP1	General large-scale benchmark MOP		√	√	√					√		√						
215	GLSMOP2	General large-scale benchmark MOP		√	√	√					√		√						
216	GLSMOP3	General large-scale benchmark MOP		√	√	√					√		√						
217	GLSMOP4	General large-scale benchmark MOP		√	√	√					√		√						
218	GLSMOP5	General large-scale benchmark MOP		√	√	√					√		√						
219	GLSMOP6	General large-scale benchmark MOP		√	√	√					√		√						
220	GLSMOP7	General large-scale benchmark MOP		√	√	√					√		√						
221	GLSMOP8	General large-scale benchmark MOP		√	√	√					√		√						
222	GLSMOP9	General large-scale benchmark MOP		√	√	√					√		√						
223	IMMOEA_F1	Benchmark MOP for testing IM-MOEA		√		√					√								
224	IMMOEA_F2	Benchmark MOP for testing IM-MOEA		√		√					√								
225	IMMOEA_F3	Benchmark MOP for testing IM-MOEA		√		√					√								
226	IMMOEA_F4	Benchmark MOP for testing IM-MOEA		√		√					√								
227	IMMOEA_F5	Benchmark MOP for testing IM-MOEA		√		√					√								
228	IMMOEA_F6	Benchmark MOP for testing IM-MOEA		√		√					√								
229	IMMOEA_F7	Benchmark MOP for testing IM-MOEA		√		√					√								
230	IMMOEA_F8	Benchmark MOP for testing IM-MOEA		√		√					√								

No.	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
231	IMMOEA_F9	Benchmark MOP for testing IM-MOEA		√		√					√								
232	IMMOEA_F10	Benchmark MOP for testing IM-MOEA		√		√					√								
233	IMOP1	Benchmark MOP with irregular Pareto front		√		√							√						
234	IMOP2	Benchmark MOP with irregular Pareto front		√		√							√						
235	IMOP3	Benchmark MOP with irregular Pareto front		√		√							√						
236	IMOP4	Benchmark MOP with irregular Pareto front		√		√							√						
237	IMOP5	Benchmark MOP with irregular Pareto front		√		√							√						
238	IMOP6	Benchmark MOP with irregular Pareto front		√		√							√						
239	IMOP7	Benchmark MOP with irregular Pareto front		√		√							√						
240	IMOP8	Benchmark MOP with irregular Pareto front		√		√							√						
241	IN1KMOP1	Neural architecture search on ImageNet 1K		√		√					√		√						
242	IN1KMOP2	Neural architecture search on ImageNet 1K		√		√					√		√						
243	IN1KMOP3	Neural architecture search on ImageNet 1K		√		√					√		√						
244	IN1KMOP4	Neural architecture search on ImageNet 1K		√		√					√		√						
245	IN1KMOP5	Neural architecture search on ImageNet 1K		√		√					√		√						
246	IN1KMOP6	Neural architecture search on ImageNet 1K		√		√					√		√						
247	IN1KMOP7	Neural architecture search on ImageNet 1K		√		√					√		√						
248	IN1KMOP8	Neural architecture search on ImageNet 1K		√		√					√		√						
249	IN1KMOP9	Neural architecture search on ImageNet 1K		√		√					√		√						
250	Instance1	Multitasking multi-objective problem (ZDT4-R + ZDT4-G)		√		√					√						√		
251	Instance2	Multitasking multi-objective problem (ZDT4-RC + ZDT4-A)		√		√					√	√					√		
252	KP	The knapsack problem	√						√		√	√							
253	LIR-CMOP1	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
254	LIR-CMOP2	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
255	LIR-CMOP3	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
256	LIR-CMOP4	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
257	LIR-CMOP5	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
258	LIR-CMOP6	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
259	LIR-CMOP7	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
260	LIR-CMOP8	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
261	LIR-CMOP9	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
262	LIR-CMOP10	Constrained benchmark MOP with large		√		√					√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		infeasible regions																	
263	LIR-CMOP11	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
264	LIR-CMOP12	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
265	LIR-CMOP13	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
266	LIR-CMOP14	Constrained benchmark MOP with large infeasible regions		√		√					√	√							
267	LRMOP1	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
268	LRMOP2	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
269	LRMOP3	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
270	LRMOP4	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
271	LRMOP5	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
272	LRMOP6	Large-scale robust multi-objective benchmark problem		√	√	√					√		√		√				√
273	LSCM1	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
274	LSCM2	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
275	LSCM3	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
276	LSCM4	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
277	LSCM5	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
278	LSCM6	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
279	LSCM7	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
280	LSCM8	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
281	LSCM9	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
282	LSCM10	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
283	LSCM11	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
284	LSCM12	Large-scale constrained multiobjective benchmark problem		√		√					√	√							
285	LSMOP1	Large-scale benchmark MOP		√	√	√					√								
286	LSMOP2	Large-scale benchmark MOP		√	√	√					√								
287	LSMOP3	Large-scale benchmark MOP		√	√	√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
288	LSMOP4	Large-scale benchmark MOP		√	√	√					√								
289	LSMOP5	Large-scale benchmark MOP		√	√	√					√								
290	LSMOP6	Large-scale benchmark MOP		√	√	√					√								
291	LSMOP7	Large-scale benchmark MOP		√	√	√					√								
292	LSMOP8	Large-scale benchmark MOP		√	√	√					√								
293	LSMOP9	Large-scale benchmark MOP		√	√	√					√								
294	MaF1	Inverted DTLZ1		√	√	√					√								
295	MaF2	DTLZ2BZ		√	√	√					√								
296	MaF3	Convex DTLZ3		√	√	√					√								
297	MaF4	Inverted and scaled DTLZ3		√	√	√					√								
298	MaF5	Scaled DTLZ4		√	√	√					√								
299	MaF6	DTLZ5IM		√	√	√					√								
300	MaF7	DTLZ7		√	√	√					√								
301	MaF8	MP-DMP		√	√	√													
302	MaF9	ML-DMP		√	√	√													
303	MaF10	WFG1		√	√	√					√								
304	MaF11	WFG2		√	√	√					√								
305	MaF12	WFG9		√	√	√					√								
306	MaF13	P7		√	√	√					√								
307	MaF14	LSMOP3		√	√	√					√								
308	MaF15	Inverted LSMOP8		√	√	√					√								
309	MaOPP_binary	Many-objective pathfinding problem based on binary encoding			√				√		√		√						
310	MaOPP_real	Many-objective pathfinding problem based on real encoding			√	√					√		√						
311	Mario	Play with Mario	√				√	√											
312	MaxCut	The max-cut problem	√						√		√								
313	MLDMP	The multi-line distance minimization problem		√	√	√													
314	MMF1	Multi-modal multi-objective test function		√		√								√					
315	MMF2	Multi-modal multi-objective test function		√		√								√					
316	MMF3	Multi-modal multi-objective test function		√		√								√					
317	MMF4	Multi-modal multi-objective test function		√		√								√					
318	MMF5	Multi-modal multi-objective test function		√		√								√					
319	MMF6	Multi-modal multi-objective test function		√		√								√					
320	MMF7	Multi-modal multi-objective test function		√		√								√					
321	MMF8	Multi-modal multi-objective test function		√		√								√					
322	MMMOP1	Multi-modal multi-objective optimization problem		√	√	√								√					
323	MMMOP2	Multi-modal multi-objective optimization problem		√	√	√								√					

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
324	MMMOP3	Multi-modal multi-objective optimization problem		√	√	√								√					
325	MMMOP4	Multi-modal multi-objective optimization problem		√	√	√								√					
326	MMMOP5	Multi-modal multi-objective optimization problem		√	√	√								√					
327	MMMOP6	Multi-modal multi-objective optimization problem		√	√	√								√					
328	MMOP_HS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
329	MMOP_HS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
330	MMOP_LS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
331	MMOP_LS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
332	MMOP_MS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
333	MMOP_MS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
334	MMOP_NS1	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
335	MMOP_NS2	Large-scale sparse multitasking multi-objective optimization problem		√		√					√				√		√		
336	MOEADDE_F1	Benchmark MOP for testing MOEA/D-DE		√		√					√								
337	MOEADDE_F2	Benchmark MOP for testing MOEA/D-DE		√		√					√								
338	MOEADDE_F3	Benchmark MOP for testing MOEA/D-DE		√		√					√								
339	MOEADDE_F4	Benchmark MOP for testing MOEA/D-DE		√		√					√								
340	MOEADDE_F5	Benchmark MOP for testing MOEA/D-DE		√		√					√								
341	MOEADDE_F6	Benchmark MOP for testing MOEA/D-DE		√		√					√								
342	MOEADDE_F7	Benchmark MOP for testing MOEA/D-DE		√		√					√								
343	MOEADDE_F8	Benchmark MOP for testing MOEA/D-DE		√		√					√								
344	MOEADDE_F9	Benchmark MOP for testing MOEA/D-DE		√		√					√								
345	MOEADM2M_F1	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
346	MOEADM2M_F2	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
347	MOEADM2M_F3	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
348	MOEADM2M_F4	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
349	MOEADM2M_F5	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
350	MOEADM2M_F6	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
351	MOEADM2M_F7	Benchmark MOP for testing MOEA/D-M2M		√		√					√								
352	MOKP	The multi-objective knapsack problem		√	√				√		√	√							
353	MONRP	The multi-objective next release problem		√					√		√								
354	MOTSP	The multi-objective traveling salesman problem		√	√					√	√								
355	MPDMP	The multi-point distance minimization problem		√	√	√													
356	mQAP	The multi-objective quadratic assignment problem		√	√					√	√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
357	MW1	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
358	MW2	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
359	MW3	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
360	MW4	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
361	MW5	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
362	MW6	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
363	MW7	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
364	MW8	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
365	MW9	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
366	MW10	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
367	MW11	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
368	MW12	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
369	MW13	Constrained benchmark MOP proposed by Ma and Wang		√		√					√	√							
370	MW14	Constrained benchmark MOP proposed by Ma and Wang		√	√	√					√	√							
371	NI_HS	Multitasking problem (Rosenbrock function + Rastrigin function)	√			√					√						√		
372	NI_MS	Multitasking problem (Griewank function + Weierstrass function)	√			√					√						√		
373	RMMEDA_F1	Benchmark MOP for testing RM-MEDA		√		√					√								
374	RMMEDA_F2	Benchmark MOP for testing RM-MEDA		√		√					√								
375	RMMEDA_F3	Benchmark MOP for testing RM-MEDA		√		√					√								
376	RMMEDA_F4	Benchmark MOP for testing RM-MEDA		√		√					√								
377	RMMEDA_F5	Benchmark MOP for testing RM-MEDA		√		√					√								
378	RMMEDA_F6	Benchmark MOP for testing RM-MEDA		√		√					√								
379	RMMEDA_F7	Benchmark MOP for testing RM-MEDA		√		√					√								
380	RMMEDA_F8	Benchmark MOP for testing RM-MEDA		√		√					√								
381	RMMEDA_F9	Benchmark MOP for testing RM-MEDA		√		√					√								
382	RMMEDA_F10	Benchmark MOP for testing RM-MEDA		√		√					√								
383	RWMOP1	Pressure vessal problem		√		√						√							
384	RWMOP2	Vibrating platform		√		√						√							
385	RWMOP3	Two bar truss design problem		√		√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
386	RWMOP4	Weldan beam design problem		√		√						√							
387	RWMOP5	Disc brake design problem		√		√						√							
388	RWMOP6	Speed reducer design problem		√		√						√							
389	RWMOP7	Gear train design problem		√		√						√							
390	RWMOP8	Car side impact design problem		√		√						√							
391	RWMOP9	Four bar plane truss		√		√						√							
392	RWMOP10	Two bar plane truss		√		√						√							
393	RWMOP11	Water resource management problem		√		√						√							
394	RWMOP12	Simply supported I-beam design		√		√						√							
395	RWMOP13	Gear box design		√		√						√							
396	RWMOP14	Multiple-disk clutch brake design problem		√		√						√							
397	RWMOP15	Spring design problem		√		√						√							
398	RWMOP16	Cantilever beam design problem		√		√						√							
399	RWMOP17	Bulk carriers design problem		√		√						√							
400	RWMOP18	Front rail design problem		√		√						√							
401	RWMOP19	Multi-product batch plant		√		√						√							
402	RWMOP20	Hydro-static thrust bearing design problem		√		√						√							
403	RWMOP21	Crash energy management for high-speed train		√		√						√							
404	RWMOP22	Haverly's pooling problem		√		√						√							
405	RWMOP23	Reactor network design		√		√						√							
406	RWMOP24	Heat exchanger network design		√		√						√							
407	RWMOP25	Process synthesis problem		√		√						√							
408	RWMOP26	Process sythesis and design problem		√		√						√							
409	RWMOP27	Process flow sheeting problem		√		√						√							
410	RWMOP28	Two reactor problem		√		√						√							
411	RWMOP29	Process synthesis problem		√		√						√							
412	RWMOP30	Synchronous pptimal pulse-width modulation of 3-level inverters		√		√						√							
413	RWMOP31	Synchronous pptimal pulse-width modulation of 5-level inverters		√		√						√							
414	RWMOP32	Synchronous pptimal pulse-width modulation of 7-level inverters		√		√						√							
415	RWMOP33	Synchronous pptimal pulse-width modulation of 9-level inverters		√		√						√							
416	RWMOP34	Synchronous pptimal pulse-width modulation of 11-level inverters		√		√						√							
417	RWMOP35	Synchronous pptimal pulse-width modulation of 13-level inverters		√		√						√							
418	RWMOP36	Optimal sizing of single phase distributed generation with reactive power support for phase balancing at main transformer/grid and active power loss		√		√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
419	RWMOP37	Optimal Sizing of Single Phase Distributed Generation with reactive power support for Phase Balancing at Main Transformer/Grid and reactive Power loss		√		√						√							
420	RWMOP38	Optimal sizing of single phase distributed generation with reactive power support for active and reactive power loss		√		√						√							
421	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		√		√						√							
422	RWMOP40	Optimal power flow for minimizing active and reactive power loss		√		√						√							
423	RWMOP41	Optimal power flow for minimizing voltage deviation, active and reactive power loss		√		√						√							
424	RWMOP42	Optimal power flow for minimizing voltage deviation, and active power loss		√		√						√							
425	RWMOP43	Optimal power flow for minimizing fuel cost, and active power loss		√		√						√							
426	RWMOP44	Optimal power flow for minimizing fuel cost, active and reactive power loss		√		√						√							
427	RWMOP45	Optimal power flow for minimizing fuel cost, voltage deviation, and active power loss		√		√						√							
428	RWMOP46	Optimal power flow for minimizing fuel cost, voltage deviation, active and reactive power loss		√		√						√							
429	RWMOP47	Optimal droop setting for minimizing active and reactive power loss		√		√						√							
430	RWMOP48	Optimal droop setting for minimizing voltage deviation and active power loss		√		√						√							
431	RWMOP49	Optimal droop setting for minimizing voltage deviation, active, and reactive power loss		√		√						√							
432	RWMOP50	Power distribution system planning		√		√						√							
433	SDC1	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
434	SDC2	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
435	SDC3	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
436	SDC4	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
437	SDC5	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
438	SDC6	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
439	SDC7	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
440	SDC8	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
441	SDC9	Scalable high-dimensional decicsion		√		√						√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
		constraint benchamrk																	
442	SDC10	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
443	SDC11	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
444	SDC12	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
445	SDC13	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
446	SDC14	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
447	SDC15	Scalable high-dimensional decicsion constraint benchamrk		√		√						√							
448	SMD1	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
449	SMD2	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
450	SMD3	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
451	SMD4	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
452	SMD5	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
453	SMD6	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
454	SMD7	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
455	SMD8	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√												√	
456	SMD9	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
457	SMD10	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
458	SMD11	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
459	SMD12	Bilevel optimization problems proposed by Sinha, Malo, and Deb		√		√						√						√	
460	SO_ISCSO_2016	International student competition in structural optimization	√				√				√	√							
461	SO_ISCSO_2017	International student competition in structural optimization	√				√				√	√							
462	SO_ISCSO_2018	International student competition in structural optimization	√				√				√	√							
463	SO_ISCSO_2019	International student competition in structural optimization	√				√				√	√							
464	SO_ISCSO_2021	International student competition in structural optimization	√				√				√	√							
465	SO_ISCSO_2022	International student competition in structural optimization	√				√				√	√							

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
466	Sparse_CD	The community detection problem		√					√		√		√		√				
467	Sparse_CN	The critical node detection problem		√					√		√		√		√				
468	Sparse_FS	The feature selection problem		√					√		√		√		√				
469	Sparse_IS	The instance selection problem		√					√		√		√		√				
470	Sparse_KP	The sparse multi-objective knapsack problem		√	√				√		√								
471	Sparse_NN	The neural network training problem		√		√					√		√		√				
472	Sparse_PM	The pattern mining problem		√					√		√		√		√				
473	Sparse_PO	The portfolio optimization problem		√		√					√		√		√				
474	Sparse_SR	The sparse signal reconstruction problem		√		√					√		√		√				
475	SMMOP1	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
476	SMMOP2	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
477	SMMOP3	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
478	SMMOP4	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
479	SMMOP5	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
480	SMMOP6	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
481	SMMOP7	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
482	SMMOP8	Sparse multi-modal multi-objective optimization problem		√	√	√					√			√	√				
483	SMOP1	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
484	SMOP2	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
485	SMOP3	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
486	SMOP4	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
487	SMOP5	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
488	SMOP6	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
489	SMOP7	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
490	SMOP8	Benchmark MOP with sparse Pareto optimal solutions		√	√	√					√		√		√				
491	SOP_F1	Sphere function	√			√							√						
492	SOP_F2	Schwefel's function 2.22	√			√							√						
493	SOP_F3	Schwefel's function 1.2	√			√							√						
494	SOP_F4	Schwefel's function 2.21	√			√							√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
495	SOP_F5	Generalized Rosenbrock's function	√			√							√						
496	SOP_F6	Step function	√			√							√						
497	SOP_F7	Quartic function with noise	√			√							√						
498	SOP_F8	Generalized Schwefel's function 2.26	√			√							√						
499	SOP_F9	Generalized Rastrigin's function	√			√							√						
500	SOP_F10	Ackley's function	√			√							√						
501	SOP_F11	Generalized Griewank's function	√			√							√						
502	SOP_F12	Generalized penalized function	√			√							√						
503	SOP_F13	Generalized penalized function	√			√							√						
504	SOP_F14	Shekel's foxholes function	√			√							√						
505	SOP_F15	Kowalik's function	√			√							√						
506	SOP_F16	Six-hump camel-back function	√			√							√						
507	SOP_F17	Branin function	√			√							√						
508	SOP_F18	Goldstein-price function	√			√							√						
509	SOP_F19	Hartman's family	√			√							√						
510	SOP_F20	Hartman's family	√			√							√						
511	SOP_F21	Shekel's family	√			√							√						
512	SOP_F22	Shekel's family	√			√							√						
513	SOP_F23	Shekel's family	√			√							√						
514	TP1	Test problem for robust multi-objective optimization		√		√					√								√
515	TP2	Test problem for robust multi-objective optimization		√		√					√								√
516	TP3	Test problem for robust multi-objective optimization		√		√					√								√
517	TP4	Test problem for robust multi-objective optimization		√		√					√								√
518	TP5	Test problem for robust multi-objective optimization		√		√					√								√
519	TP6	Test problem for robust multi-objective optimization		√		√					√								√
520	TP7	Test problem for robust multi-objective optimization		√		√					√								√
521	TP8	Test problem for robust multi-objective optimization		√		√					√								√
522	TP9	Test problem for robust multi-objective optimization		√		√					√								√
523	TP10	Test problem for robust multi-objective optimization		√		√					√	√							√
524	TREE1	The time-varying ratio error estimation problem		√		√					√	√	√						
525	TREE2	The time-varying ratio error estimation problem		√		√					√	√	√						
526	TREE3	The time-varying ratio error estimation problem		√		√					√	√	√						
527	TREE4	The time-varying ratio error estimation problem		√		√					√	√	√						
528	TREE5	The time-varying ratio error estimation problem		√		√					√	√	√						
529	TREE6	The time-varying ratio error estimation problem		√		√					√	√	√						
530	TSP	The traveling salesman problem	√							√	√								
531	UF1	Unconstrained benchmark MOP		√		√					√								

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
532	UF2	Unconstrained benchmark MOP		√		√					√								
533	UF3	Unconstrained benchmark MOP		√		√					√								
534	UF4	Unconstrained benchmark MOP		√		√					√								
535	UF5	Unconstrained benchmark MOP		√		√					√								
536	UF6	Unconstrained benchmark MOP		√		√					√								
537	UF7	Unconstrained benchmark MOP		√		√					√								
538	UF8	Unconstrained benchmark MOP		√		√					√								
539	UF9	Unconstrained benchmark MOP		√		√					√								
540	UF10	Unconstrained benchmark MOP		√		√					√								
541	VNT1	Benchmark MOP proposed by Viennet		√		√													
542	VNT2	Benchmark MOP proposed by Viennet		√		√													
543	VNT3	Benchmark MOP proposed by Viennet		√		√													
544	VNT4	Benchmark MOP proposed by Viennet		√		√						√							
545	WFG1	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
546	WFG2	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
547	WFG3	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
548	WFG4	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
549	WFG5	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
550	WFG6	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
551	WFG7	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
552	WFG8	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
553	WFG9	Benchmark MOP proposed by Walking Fish Group		√	√	√					√		√						
554	ZCAT1	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
555	ZCAT2	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
556	ZCA3	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
557	ZCA4	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
558	ZCA5	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
559	ZCAT6	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
560	ZCAT7	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
561	ZCAT8	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
562	ZCAT9	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
563	ZCAT10	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						

	Abbreviation	Full name	single	multi	many	real	integer	label	binary	permutation	large	constrained	expensive	multimodal	sparse	dynamic	multitask	bilevel	robust
564	ZCAT11	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
565	ZCAT12	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
566	ZCAT13	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
567	ZCAT14	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
568	ZCAT15	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
569	ZCAT16	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
570	ZCAT17	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
571	ZCAT18	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
572	ZCAT19	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
573	ZCAT20	Benchmark MOP proposed by Zapotecas, Coello, Aguirre, and Tanaka		√	√	√					√		√						
574	ZDT1	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
575	ZDT2	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
576	ZDT3	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
577	ZDT4	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
578	ZDT5	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√					√		√		√						
579	ZDT6	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√					√		√						
580	ZXH_CF1	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
581	ZXH_CF2	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
582	ZXH_CF3	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
583	ZXH_CF4	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
584	ZXH_CF5	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
585	ZXH_CF6	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
586	ZXH_CF7	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
587	ZXH_CF8	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
588	ZXH_CF9	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
		Zhou, Xiang, and He																	
589	ZXH_CF10	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
590	ZXH_CF11	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
591	ZXH_CF12	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
592	ZXH_CF13	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
593	ZXH_CF14	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
594	ZXH_CF15	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							
595	ZXH_CF16	Constrained benchmark MOP proposed by Zhou, Xiang, and He		√	√	√					√	√							