PlatEMO

Evolutionary Multi-Objective Optimization Platform

User Manual 3.3

BIMK Group

August 14, 2021

Thank you very much for using PlatEMO. The copyright of PlatEMO belongs to the BIMK Group. 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 reference the following literature:

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.

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

Contents

I. Quick Start

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

PlatEMO provides a variety of algorithms for solving optimization problems in a black-box manner. To this end, users should define the optimization problem, select an algorithm, and set the parameter values, by means of one of the following ways:

1) Calling the main function with parameters:

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

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:

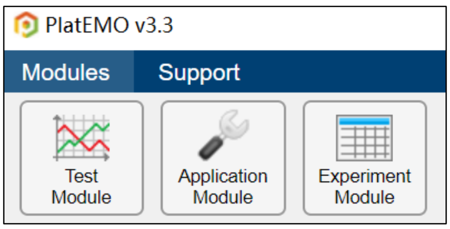
```
f1 = @(x,d)sum(x*d);
f2 = @(x,d)1-sum(x*d);
platemo('objFcn',f1,'conFcn',f2,'algorithm',@GA,…);
```

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

3) Calling the main function without parameter:

```
platemo();
```

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



II. Using PlatEMO without GUI

A. Solving Benchmark Problems

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

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

where all the acceptable names and values are

Name	Data type	Default value	Description
`'algorithm'`	Function handle or cell	`dependent`	Class of algorithm
`'problem'`	Function handle or cell	`dependent`	Class of benchmark 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`	Number of evaluations
`'save'`	Integer	`0`	Number of saved populations
`'outputFcn'`	Function handle	`@ALGORITHM.Output`	Function called before each iteration

- `'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.
- `'problem'` denotes the benchmark problem to be solved, 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.
- `'N'` denotes the population size of the algorithm, which usually equals to the number of solutions in the final population.
- `'M'` denotes the number of objectives of the benchmark problem, which is valid for some multi-objective benchmark problems.
- `'D'` denotes the number of decision variables of the benchmark problem, which is valid for some benchmark problems.
- `'maxFE'` denotes the maximum number of available function evaluations, which usually equals to the product of population size and number of generations.
- `'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 zero (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.

For example, the following code runs the genetic algorithm on the sphere function with a population size of 50, where the populations are displayed in a figure:

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

The following code runs NSGA-II on 5-objective 40-variable DTLZ2 for 20000 function evaluations, where the populations are saved to a file:

```
platemo('algorithm',@NSGAII,'problem',@DTLZ2,'M',5,'D',40,'
maxFE',20000,'save',10);
```

The following code runs MOEA/D with Tchebycheff approach on ZDT1 for ten times, where the populations obtained in each time are saved to a file:

```
for i = 1 : 10
    platemo('algorithm',{@MOEAD,2},'problem',@ZDT1,'save',5);
end
```

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

B. Solving User-Defined Problems

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

Name	Data type	Default value	Description
`'encoding'`	char	`'real'`	Encoding scheme
`'objFcn'`	Function handle or cell	`@(x,d)sum(x)`	Objective functions. All objectives are to be minimized
`'conFcn'`	Function handle or cell	`@(x,d)0`	Constraint functions. A constraint is satisfied if and only if the constraint violation is not positive
`'lower'`	Row vector	0	Lower bounds of variables
`'upper'`	Row vector	1	Upper bounds of variables
`'initFcn'`	Function handle	`[]`	Function for initializing a population
`'decFcn'`	Function handle	`[]`	Function for repairing invalid solution
`'parameter'`	Cell	`{}`	Data

- `'encoding'` denotes the encoding scheme of the problem, whose value can be 'real' (variables are real or integer numbers), 'binary' (variables are binary numbers), or 'permutation' (variables constitute a permutation). Algorithms may use different reproduction operators for different encoding schemes.

- 'objFcn' denotes the objective functions of the problem, whose value can be a function handle (a single objective) or cell (multiple objectives). An objective function has two inputs and an output, where the first input is a decision vector, the second input is the data specified by 'parameter' (this input can be omitted if 'parameter' is not specified), and the output is the objective value. All the objectives are to be minimized.
- 'conFcn' denotes the constraint functions of the problem, whose value can be a function handle (a single constraint) or cell (multiple constraints). A constraint function has two inputs and an output, where the first input is a decision vector, the second input is the data specified by 'parameter' (this input can be omitted if 'parameter' is not specified), and the output is the constraint violation. A constraint is satisfied if and only if the constraint violation is not positive.
- 'lower' denotes the lower bounds of variables, which is effective only when the value of 'encoding' is 'real'.
- 'upper' denotes the upper bounds of variables, which is effective only when the value of 'encoding' is 'real'.
- 'initFcn' denotes the function for initializing a population, whose value should be a function handle having two inputs and an output, where the first input is the number of solutions in the population, the second input is the data specified by 'parameter' (this input can be omitted if 'parameter' is not specified), and the output is a matrix consisting of the decision vectors in the initial population. This function is called at the beginning of most algorithms.
- 'decFcn' denotes the function for repairing invalid solution, whose value should be a function handle having two inputs and an output, where the first input is a decision vector, the second input is the data specified by 'parameter' (this input can be omitted if 'parameter' is not specified), and the output is the repaired decision vector. This function is called before the objective calculation of each solution.
- 'parameter' denotes the data of the problem, which is used as the second input of the functions specified by 'objFcn', 'conFcn', 'initFcn', and 'decFcn'.

For example, the following code solves a unimodal problem with 10 variables by differential evolution:

```
platemo('objFcn',@(x)sum(x.^2),'lower',zeros(1,10)-10,
'upper',zeros(1,10)+10,'algorithm',@DE);
```

The following code solves a rotated unimodal problem with 10 variables by the default algorithm, where the rotation matrix is specified via `'parameter'`:

```
platemo('objFcn',@(x,d)sum((x*d).^2),'lower',zeros(1,10)-10,'upper',zeros(1,10)+10,'parameter',rand(10));
```

The following code solves a constrained bi-objective problem with 20 variables by NSGA-II with a population size of 50:

```
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},'conFcn',g1,'lower',zeros(1,20),'upper',ones(1,20),'algorithm',@NSGAII,'N',50);
```

C. Collecting the Results

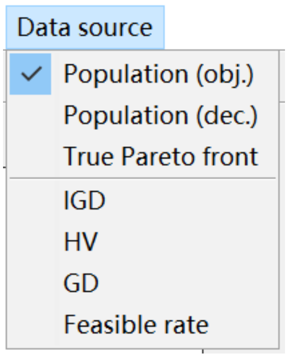
The generated populations can be displayed, saved, or returned after the algorithm terminates. If the main function is called like

```
[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 zero (default), where various plots can be displayed by switching the `Data source` menu on the figure.



While if `Value` is positive, the generated populations will be saved to a MAT file named as `PlatEMO\Data\alg\alg_pro_M_D_run.mat`, where `alg` is the algorithm name, `pro` is the problem name, `M` is the number of objectives, `D` is the number of variables, and `run` automatically increases from 1 until the file name does not exist. A file saves a cell `result` consisting of the generated populations and a struct `metric` consisting of the metric values. The whole optimization process of the algorithm is divided into `Value` equal intervals, where the first column of `result` stores the number of consumed function evaluations at the last iteration of each interval, the second column of `result` stores the population at the last iteration of each interval, and `metric` stores the metric values of the stored populations. Note that the above are achieved by the default output function `@ALGORITHM.Output`, while users can collect the results in their own ways by specifying the value of `'outputFcn'` to the handle of a user-defined output function.

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

```
metric =
  struct with fields:

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

Besides, the metric values can be automatically calculated and saved in the experiment module of the GUI. To calculate the metric values manually, users should obtain the optimums of the problem and then call the metric functions, for example,

```
pro = DTLZ2();
IGD(result{end},pro.optimum);
```

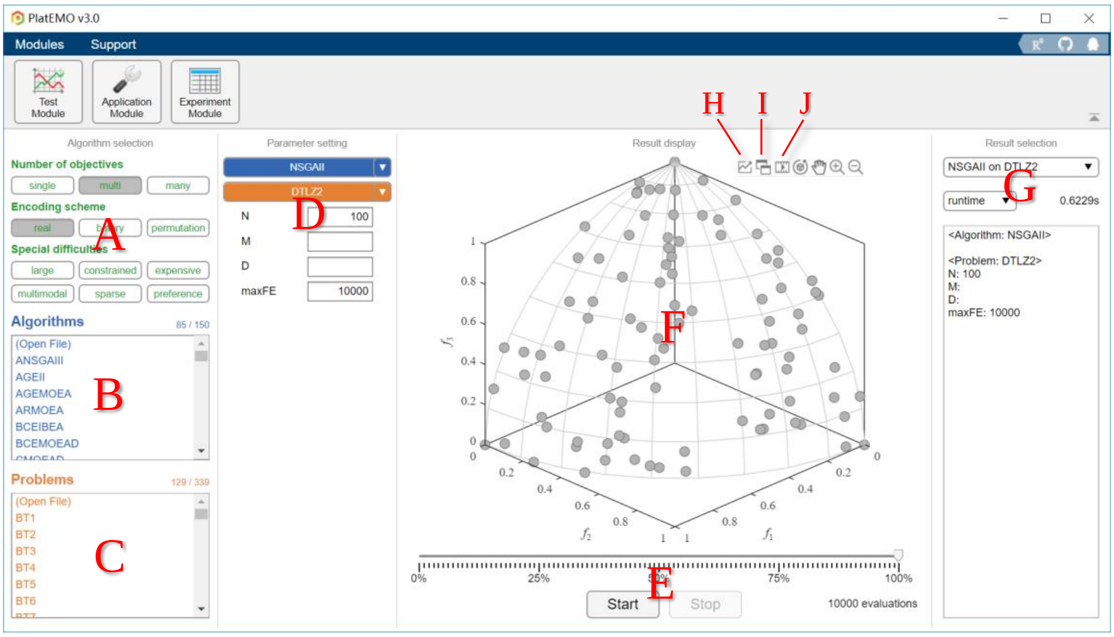
III. Using PlatEMO with GUI

A. Functions of Test Module

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

```
platemo();
```

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

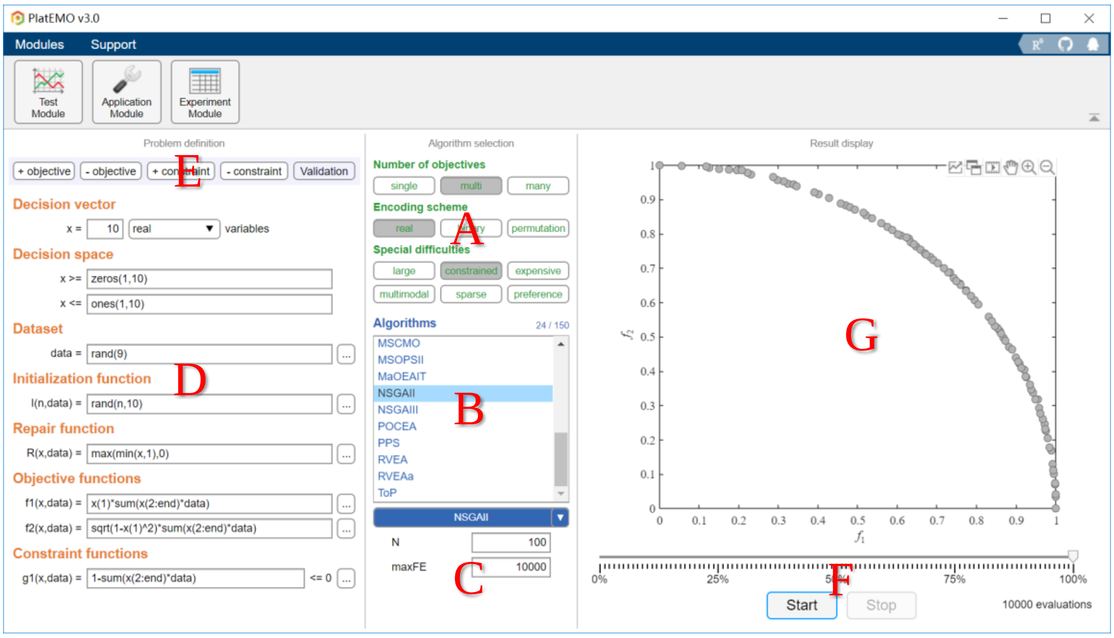


Users should first determine the type of problems in Region A (see *Labels of Algorithms and Problems* for details), select an algorithm in Region B, select a benchmark problem in Region C, and set the parameter values in Region D. Then, the optimization process can be started and controlled in Region E, where the real-time result is displayed in Region F and the historical results can be reviewed in Region G.

Pressing Button H can choose the plot to be displayed, pressing Button I can display the plot in a new figure and save the data in the plot to workspace, and pressing Button J can save the whole optimization process to a GIF file with 20 frames.

B. Functions of Application Module

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



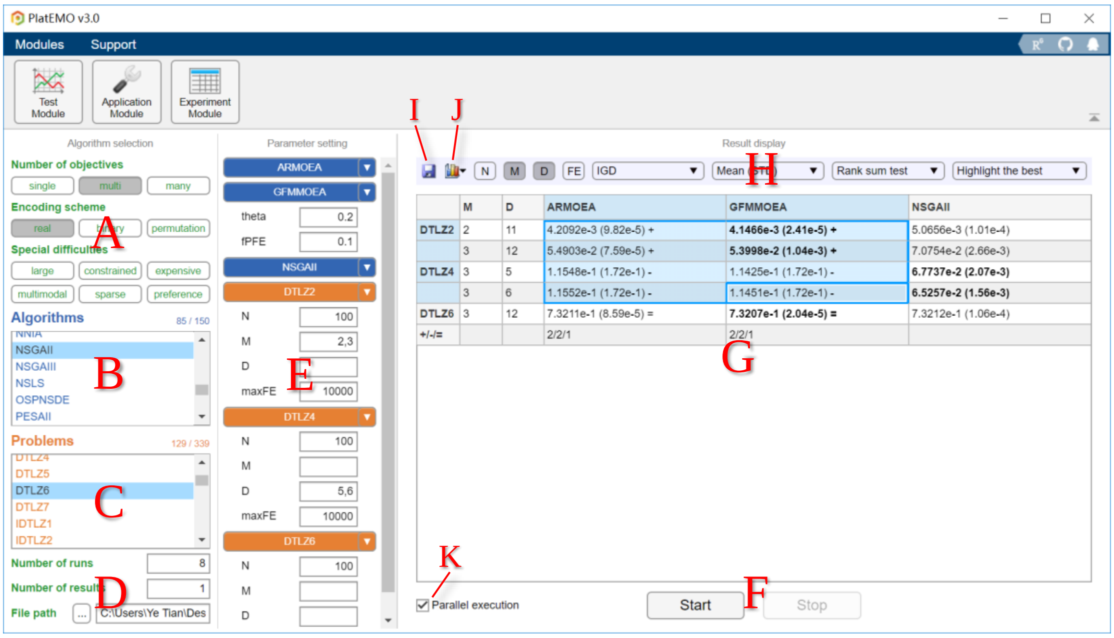
Users should first define the problem in Region D, whose details are the same to those in *Solving User-Defined Problems*, where

- Decision vector is the same to `'encoding'`
- Decision space is the same to `'lower'` and `'upper'`
- Dataset is the same to `'parameter'`
- Initialization function is the same to `'initFcn'`
- Repair function is the same to `'decFcn'`
- Objective functions is the same to `'objFcn'`
- Constraint functions is the same to `'conFcn'`

For simplicity, users can only specify Decision vector, Decision space, Objective functions, and Constraint functions. Meanwhile, users can increase or decrease the numbers of objectives and constraints, and check the validity of the problem in Region E. Then, the type of problems can be automatically determined in Region A, while users should select an algorithm in Region B and set the parameter values in Region C. The optimization process can be started and controlled in Region F, and the real-time result is displayed in Region G.

C. Functions of Experiment Module

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



Users should first determine the type of problems in Region A (see *Labels of Algorithms and Problems* for details), select multiple algorithms in Region B, select multiple benchmark problems in Region C, configure the experimental settings in Region D, and set the parameter values in Region E, where the number of objectives `M` and the number of variables `D` can be vectors. Then, the optimization process can be started and controlled in Region F, where the statistical results are listed in Region G.

The statistical results to be listed can be customized in Region H. Pressing Button I can save the table to an Excel, TeX, TXT, or MAT file, and pressing Button J can display the results in the selected cells of the table in a new figure. Button K determines whether the experiment is performed on a single CPU (in sequence) or all the CPUs (in parallel).

All the results are saved to MAT files in the folder specified in Region D. If a result file already exists, the file will be loaded and the algorithm will not be run.

D. Labels of Algorithms and Problems

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

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

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

Label	Description
<single>	The problem has a single objective
<multi>	The problem has two or three objectives
<many>	The problem has four or more objectives
<real>	The decision variables are real or integer numbers
<binary>	The decision variables are binary numbers
<permutation>	The decision variables constitute a permutation
<large>	The problem has more than 100 decision variables
<constrained>	The problem has at least one constraint
<expensive>	The objectives are computationally expensive, i.e., only a very limited number of function evaluations are available
<multimodal>	There exist multiple optimal solutions with similar objective values but considerably different decision vectors, all of which should be found
<sparse>	Most decision variables of the optimal solutions are zero
<preference>	Only the optimal solutions in the predefined regions of the Pareto front are expected to be found
<none>	Empty label

An algorithm may have multiple sets of labels, where the Cartesian product between all the label sets include 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 or benchmark problem should be tagged with labels, otherwise it will not be appeared in the lists in the GUI. When determining the type of problems in Region A, the algorithms that can solve such type of problems will be appeared in the list in Region B, and the benchmark problems belonging to this type will be appeared in the list in Region C. The labels of all the algorithms and benchmark problems in PlatEMO are referred to *List of Algorithms* and *List of Problems*, respectively.

IV. Extending PlatEMO

A. ALGORITHM Class

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

Property	Specified by	Description
`parameter`	Users	Parameters of the algorithm
`save`	Users	Number of populations saved in an execution
`outputFcn`	Users	Function called in `NotTerminated()`
`pro`	`Solve()`	Problem solved in current execution
`result`	`NotTerminated()`	Populations saved in current execution
`metric`	`NotTerminated()`	Metric values of current populations
Method	**Be redefined**	**Description**
`ALGORITHM`	Cannot	Set the properties specified by users
`Solve`	Cannot	Call `alg.Solve(pro)` to solve problem `pro` by algorithm `alg`
`main`	Must	Main procedure of the algorithm
`NotTerminated`	Cannot	Function called before each iteration in `main()`
`ParameterSet`	Cannot	Set the parameter values according to `parameter`

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

```
classdef GA < ALGORITHM
% <single><real/binary/permutation><large/none><constrained/none>
% Genetic algorithm
% proC --- 1 --- Probability of crossover
% disC --- 20 --- Distribution index of crossover
% proM --- 1 --- Expectation of the number of mutated variables
% disM --- 20 --- Distribution index of mutation

%------------------------------- Reference --------------------------------
% J. H. Holland, Adaptation in Natural and Artificial Systems,
% MIT Press, 1992.
%--------------------------------------------------------------------------

    methods
        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                  P1 = TournamentSelection(2,Pro.N,FitnessSingle(P));
20                  O = OperatorGA(P(P1),{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
```

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 and Problems* 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 last population and checking whether the number of function evaluations exceeds; if so, the algorithm will terminate immediately;
Line 19:	Binary tournament based mating selection by calling a public function;
Line 20:	Using the mating pool to generate offsprings by calling a public function;
Line 21:	Combing the current population with the offsprings;
Line 22:	Sorting the solutions based on their fitness calculated by a public function;
Line 23:	Retaining the solutions with better fitness for 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`, which provides a number of operations commonly used in algorithms. 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
ALGORITHM. ParameterSet	Set the parameter values specified by users
PROBLEM. Initialization	Initialize a population for the problem
CrowdingDistance	Crowding distance calculation for multi-objective optimization
FitnessSingle	Fitness calculation for single-objective optimization
NDSort	Non-dominated sorting
OperatorDE	The reproduction operator of differential evolution
OperatorFEP	The reproduction operator of fast evolutionary programming
OperatorGA	The reproduction operators of genetic algorithm
OperatorGAhalf	The reproduction operators of genetic algorithm, where only the first half of offsprings are generated
OperatorPSO	The reproduction operator of particle swarm optimization
RouletteWheel Selection	Roulette-wheel selection
Tournament Selection	Tournament selection
UniformPoint	Generate a set of uniformly distributed points

B. PROBLEM Class

A benchmark 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	SOLUTION()	Number of function evaluations consumed in current execution
encoding	Setting()	Encoding scheme of the problem
lower	Setting()	Lower bounds of the decision variables
upper	Setting()	Upper bounds of the decision variables
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
`Setting`	Must	Default settings of the problem
`Initialization`	Can	Initialize a population for the problem
`CalDec`	Can	Repair invalid solutions in a population
`CalObj`	Must	Calculate the objective values of solutions in a population. All objectives are to be minimized
`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
`GetOptimum`	Can	Generate the optimal values and store in `optimum`
`GetPF`	Can	Generate the Pareto front and store in `PF`
`DrawDec`	Can	Display the decision variables of a population
`DrawObj`	Can	Display the objective values of a population
`Current`	Cannot	Static method for getting or setting the current `PROBLEM` object
`ParameterSet`	Cannot	Set the parameter values according to `parameter`

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

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

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

    methods
        function Setting(obj)
            obj.M = 1;
            if isempty(obj.D); obj.D = 30; end
            obj.lower = zeros(1,obj.D) - 100;
            obj.upper = zeros(1,obj.D) + 100;
            obj.encoding = 'real';
        end
        function PopObj = CalObj(obj,PopDec)
            PopObj = sum(PopDec.^2,2);
        end
    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 and Problems* 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 scheme of the problem;
Line 19:	Redefining the method of calculating objective values;
Line 20:	Calculating the objective values of solutions in a population.

The method `Initialization()` randomly initializes a population for the problem. 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 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:

```
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 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, `MW1.m` calculates a constraint for each solution:

```
function PopCon = CalCon(obj,X)
    PopObj = obj.CalObj(X);
    l = sqrt(2)*PopObj(:,2) - sqrt(2)*PopObj(:,1);
    PopCon = sum(PopObj,2) - 1 - 0.5*sin(2*pi*l).^8;
end
```

Use `all(PopCon<=0,2)` to determine whether each solution is feasible or not. Note that equality constraints should be relaxed to such inequality constraints. The method `GetOptimum()` can be redefined to specify the optimal values of the problem. 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 the problem for the visualization achieved in `DrawObj()`. For example, `DTLZ2.m` returns the data for plotting the 2-D or 3-D Pareto front:

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

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

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

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

where `Draw()` is a function in the folder `PlatEMO\GUI` for displaying data. The details of the above functions are referred to the comments in their codes.

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`	Receive the decision variables and calculate the objective values and constraint violations of one or more solutions. `PROBLEM.FE` will be automatically increased by the number of `SOLUTION` objects returned	
`decs`	Get the matrix of decision variables of multiple solutions	
`objs`	Get the matrix of objective values of multiple solutions	
`cons`	Get the matrix of constraint violations of multiple solutions	
`adds`	Get the matrix of additional properties of multiple solutions	
`best`	Get the feasible and best solution for single-objective optimization, or the feasible and non-dominated solutions for multi-objective optimization	

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));
BestObjs = Population.best.objs
```

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.pro    = Problem;
    obj.result = {};
    obj.metric = struct('runtime',0);
    obj.pro.FE = 0;
    addpath(fileparts(which(class(obj))));
    addpath(fileparts(which(class(obj.pro))));
    PROBLEM.Current(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(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
  switch obj.encoding
    case 'binary'
      PopDec = randi([0,1],N,obj.D);
    case 'permutation'
      [~,PopDec] = sort(rand(N,obj.D),2);
    otherwise
      PopDec = unifrnd(repmat(obj.lower,N,1),repmat(obj.upper,N,1));
  end
  Population = SOLUTION(PopDec);
end
```

SOLUTION.SOLUTION()

```
function obj = SOLUTION(PopDec,AddPro)
  if nargin > 0
    obj(1,size(PopDec,1)) = SOLUTION;
    Problem = PROBLEM.Current();
    PopDec  = Problem.CalDec(PopDec);
    PopObj  = Problem.CalObj(PopDec);
    PopCon  = Problem.CalCon(PopDec);
    for i = 1 : length(obj)
      obj(i).dec = PopDec(i,:);
      obj(i).obj = PopObj(i,:);
      obj(i).con = PopCon(i,:);
    end
    if nargin > 1
      for i = 1 : length(obj)
        obj(i).add = AddPro(i,:);
      end
    end
    Problem.FE = Problem.FE + length(obj);
  end
end
```

PROBLEM.Current()

```
function obj = Current(obj)
  persistent Problem;
  if nargin > 0; Problem = obj; end
  if nargout > 0; obj = Problem; end
end
```

ALGORITHM.Output()

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

ALGORITHM.NotTerminated()

```
function nofinish = NotTerminated(obj,Population)
  obj.metric.runtime = obj.metric.runtime + toc;
  if obj.save <= 0; num = 10; else; num = obj.save; end
  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(); 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
```

TournamentSelection()

```
function index = TournamentSelection(K,N,varargin)
  f = @(S)reshape(S,[],1);
  varargin = cellfun(f,varargin,'UniformOutput',false);
  [~,rank] = sortrows([varargin{:}]);
  [~,rank] = sort(rank);
  Parents  = randi(length(varargin{1}),K,N);
  [~,best] = min(rank(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(Parent,Parameter)
  ... ...
  if calObj
    Offspring = SOLUTION(Offspring);
  end
end
```

PROBLEM.CalDec()

```
function PopDec = CalDec(obj,PopDec)
  if strcmp(obj.encoding,'real')
    N = size(PopDec,1);
    PopDec = min(PopDec,repmat(obj.upper,N,1));
    PopDec = max(PopDec,repmat(obj.lower,N,1));
  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

```
1  function score = IGD(Population,optimum)
2  % <min>
3  % Inverted generational distance
4
5  %-------------------------- Reference ------------------------
6  % C. A. Coello Coello and N. C. Cortes, Solving multiobjective
7  % optimization problem using an artificial immune system, Genetic
8  % Programming and Evolvable Machines, 2005, 6(2): 163-190.
9  %------------------------------------------------------------------
10
11     PopObj = Population.best.objs;
12     if size(PopObj,2) ~= size(optimum,2)
13       score = nan;
14     else
15       score = mean(min(pdist2(optimum,PopObj),[],2));
16     end
17 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 `<min>` (the smaller metric value the better) or `<max>` (the larger metric value the better);

Line 3: Full name of the metric;

Lines 5-9: Reference of the metric;

Line 11: Obtaining the feasible and non-dominated solutions in the population;

Lines 12-13: Returns `nan` if there is no feasible and non-dominated solution;

Lines 14-15: Returns the IGD value of the feasible and non-dominated solutions.

V. List of Algorithms

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
1	ABC	Artificial bee colony algorithm	√			√			√	√				
2	AB-SAEA	Adaptive Bayesian based surrogate-assisted evolutionary algorithm		√	√	√					√			
3	ACO	Ant colony optimization	√					√	√					
4	AGE-II	Approximation-guided evolutionary multi-objective algorithm II		√		√	√	√						
5	AGE-MOEA	Adaptive geometry estimation-based many-objective evolutionary algorithm		√	√	√	√	√		√				
6	A-NSGA-III	Adaptive NSGA-III		√	√	√	√	√		√				
7	AR-MOEA	Adaptive reference points based multi-objective evolutionary algorithm		√	√	√	√	√		√				
8	BCE-IBEA	Bi-criterion evolution based IBEA		√	√	√	√	√						
9	BCE-MOEA/D	Bi-criterion evolution based MOEA/D		√	√	√	√	√						
10	BFGS	A quasi-Newton method proposed by Broyden, Fletcher, Goldfarb, and Shanno	√			√			√					
11	BiGE	Bi-goal evolution			√	√	√	√						
12	BSPGA	Binary space partition tree based genetic algorithm	√				√		√	√				
13	CA-MOEA	Clustering based adaptive multi-objective evolutionary algorithm		√		√	√	√						
14	CCGDE3	Cooperative coevolution GDE3		√		√			√					
15	CCMO	Coevolutionary constrained multi-objective optimization framework		√		√	√	√		√				
16	c-DPEA	Constrained dual-population evolutionary algorithm		√		√	√	√		√				
17	CMA-ES	Covariance matrix adaptation evolution strategy	√			√			√	√				
18	C-MOEA/D	Constraint-MOEA/D		√	√	√	√	√		√				
19	CMOEA-MS	Constrained multiobjective evolutionary algorithm with multiple stages		√		√	√	√		√				
20	CMOPSO	Competitive mechanism based multi-objective particle swarm optimizer		√		√								
21	CPS-MOEA	Classification and Pareto domination based multi-objective evolutionary		√		√					√			
22	CSEA	Classification based surrogate-assisted evolutionary algorithm		√	√	√					√			
23	CSO	Competitive swarm optimizer	√			√			√	√				
24	C-TAEA	Two-archive evolutionary algorithm for constrained MOPs		√	√	√	√	√		√				
25	DAEA	Duplication analysis based evolutionary algorithm		√			√							
26	DCNSGA-III	Dynamic constrained NSGA-III		√	√	√	√	√		√				
27	DE	Differential evolution	√			√			√	√				
28	DEA-GNG	Decomposition based evolutionary algorithm guided by growing neural gas		√	√	√	√	√						

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
29	DGEA	Direction guided evolutionary algorithm		√	√	√			√					
30	DMOEA-eC	Decomposition-based multi-objective evolutionary algorithm with the e-constraint framework		√		√	√	√						
31	dMOPSO	MOPSO based on decomposition		√		√								
32	DN-NSGA-II	Decision space based niching NSGA-II		√		√						√		
33	DWU	Dominance-weighted uniformity multi-objective evolutionary algorithm		√		√	√	√						
34	EAG-MOEA/D	External archive guided MOEA/D		√		√	√	√						
35	EDN-ARMOEA	Efficient dropout neural network based AR-MOEA		√	√	√					√			
36	EFR-RR	Ensemble fitness ranking with a ranking restriction scheme		√	√	√	√	√						
37	EGO	Efficient global optimization	√			√					√			
38	EIM-EGO	Expected improvement matrix based efficient global optimization		√		√					√			
39	e-MOEA	Epsilon multi-objective evolutionary algorithm		√	√	√	√	√						
40	EMyO/C	Evolutionary many-objective optimization algorithm with clustering-based		√	√	√								
41	ENS-MOEA/D	Ensemble of different neighborhood sizes based MOEA/D		√	√	√								
42	FEP	Fast evolutionary programming	√			√			√	√				
43	FRCG	Fletcher-Reeves conjugate gradient	√			√			√					
44	FROFI	Feasibility rule with the incorporation of objective function information	√			√			√	√				
45	GA	Genetic algorithm	√			√	√	√	√	√				
46	GDE3	Generalized differential evolution 3		√		√				√				
47	GFM-MOEA	Generic front modeling based multi-objective evolutionary algorithm		√	√	√	√	√						
48	GLMO	Grouped and linked mutation operator algorithm		√		√			√					
49	g-NSGA-II	g-dominance based NSGA-II		√		√	√	√						√
50	GrEA	Grid-based evolutionary algorithm			√	√	√	√						
51	HeE-MOEA	Multiobjective evolutionary algorithm with heterogeneous ensemble based infill criterion		√		√					√			
52	hpaEA	Hyperplane assisted evolutionary algorithm		√	√	√	√	√						
53	HypE	Hypervolume estimation algorithm		√	√	√	√	√						
54	IBEA	Indicator-based evolutionary algorithm		√	√	√	√	√						
55	ICMA	Indicator based constrained multi-objective algorithm		√		√				√				
56	I-DBEA	Improved decomposition-based evolutionary algorithm		√	√	√	√	√		√				
57	IM-MOEA	Inverse modeling based multiobjective evolutionary algorithm		√		√			√					
58	IMODE	Improved multi-operator differential evolution	√			√			√	√				
59	I-SIBEA	Interactive simple indicator-based evolutionary algorithm		√		√	√	√						√
60	KnEA	Knee point driven evolutionary algorithm			√	√	√	√		√				
61	K-RVEA	Surrogate-assisted RVEA		√	√	√					√			
62	KTA2	Kriging-assisted Two_Arch2		√	√	√					√			

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
63	LCSA	Linear combination-based search algorithm		√	√	√			√					
64	LMEA	Evolutionary algorithm for large-scale many-objective optimization		√	√	√			√					
65	LMOCSO	Large-scale multi-objective competitive swarm optimization algorithm		√	√	√			√	√				
66	LSMOF	Large-scale multi-objective optimization framework with NSGA-II		√		√			√					
67	MaOEA-CSS	Many-objective evolutionary algorithms based on coordinated selection		√	√	√	√	√						
68	MaOEA-DDFC	Many-objective evolutionary algorithm based on directional diversity and favorable convergence		√	√	√	√	√						
69	MaOEA/IGD	IGD based many-objective evolutionary algorithm			√	√	√	√						
70	MaOEA/IT	Many-objective evolutionary algorithms based on an independent two-stage		√	√	√				√				
71	MaOEA-R&D	Many-objective evolutionary algorithm based on objective space reduction			√	√	√	√						
72	MMOPSO	MOPSO with multiple search strategies		√		√								
73	MO_Ring_ PSO_SCD	Multiobjective PSO using ring topology and special crowding distance		√		√						√		
74	MOCell	Cellular genetic algorithm		√		√	√	√		√				
75	MO-CMA	Multi-objective covariance matrix adaptation evolution strategy		√		√								
76	MOEA/D	Multiobjective evolutionary algorithm based on decomposition		√	√	√	√	√						
77	MOEA/D-AWA	MOEA/D with covariance matrix adaptation evolution strategy		√	√	√								
78	MOEA/D-CMA	MOEA/D with covariance matrix adaptation evolution strategy		√	√	√								
79	MOEA/DD	Many-objective evolutionary algorithm based on dominance and decomposition		√	√	√	√	√		√				
80	MOEA/D-DAE	MOEA/D with detect-and-escape strategy		√		√	√	√		√				
81	MOEA/D-DE	MOEA/D based on differential evolution		√	√	√								
82	MOEA/D-DRA	MOEA/D with dynamical resource allocation		√	√	√								
83	MOEA/D-DU	MOEA/D with a distance based updating strategy		√	√	√	√	√						
84	MOEA/D-EGO	MOEA/D with efficient global optimization		√		√					√			
85	MOEA/D-FRRMAB	MOEA/D with fitness-rate-rank-based multiarmed bandit		√	√	√								
86	MOEA/D-M2M	MOEA/D based on MOP to MOP		√		√								
87	MOEA/D-MRDL	MOEA/D with maximum relative diversity loss		√		√								
88	MOEA/D-PaS	MOEA/D with Pareto adaptive scalarizing approximation		√	√	√								
89	MOEA/D-STM	MOEA/D with stable matching		√	√	√								
90	MOEA/D-URAW	MOEA/D with uniform randomly adaptive weights		√	√	√	√	√						
91	MOEA/DVA	Multi-objective evolutionary algorithm based on decision variable		√		√			√					

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
92	MOEA/IGD-NS	Multi-objective evolutionary algorithm based on an enhanced IGD		√		√	√	√						
93	MOEA-PC	Multiobjective evolutionary algorithm based on polar coordinates		√		√								
94	MOEA/PSL	Multi-objective evolutionary algorithm based on Pareto optimal subspace		√		√	√		√	√			√	
95	MOMBI-II	Many objective metaheuristic based on the R2 indicator II		√	√	√	√	√						
96	MOPSO	Multi-objective particle swarm optimization		√		√								
97	MOPSO-CD	MOPSO with crowding distance		√		√								
98	M-PAES	Memetic algorithm with Pareto archived evolution strategy		√		√								
99	MP-MMEA	Multi-population multi-modal multi-objective evolutionary algorithm		√		√			√			√	√	
100	MPSO/D	Multi-objective particle swarm optimization algorithm based on decomposition		√	√	√								
101	MSCMO	Multi-stage constrained multi-objective evolutionary algorithm		√		√	√	√		√				
102	MSEA	Multi-stage multi-objective evolutionary algorithm		√		√	√	√						
103	MSOPS-II	Multiple single objective Pareto sampling II		√	√	√				√				
104	MTS	Multiple trajectory search		√		√								
105	MultiObjective EGO	Multi-objective efficient global optimization		√		√				√	√			
106	MyO-DEMR	Many-objective differential evolution with mutation restriction		√	√	√								
107	NMPSO	Novel multi-objective particle swarm optimization		√	√	√								
108	NNIA	Nondominated neighbor immune algorithm		√		√	√	√						
109	NSGA-II	Nondominated sorting genetic algorithm II		√		√	√	√		√				
110	NSGA-II+ARSBX	NSGA-II with adaptive rotation based simulated binary crossover		√		√				√				
111	NSGA-II-conflict	NSGA-II with conflict-based partitioning strategy			√	√	√	√						
112	NSGA-III	Nondominated sorting genetic algorithm III		√	√	√	√	√		√				
113	NSGA-II/SDR	NSGA-II with strengthened dominance relation			√	√	√	√						
114	NSLS	Multiobjective optimization framework based on nondominated sorting and local search		√		√								
115	OFA	Optimal foraging algorithm	√			√			√	√				
116	one-by-one EA	Many-objective evolutionary algorithm using a one-by-one selection		√	√	√	√	√						
117	OSP-NSDE	Non-dominated sorting differential evolution with prediction in the objective space		√		√								
118	ParEGO	Efficient global optimization for Pareto optimization		√		√					√			
119	PeEA	Pareto front shape estimation based evolutionary algorithm		√	√	√	√	√						
120	PESA-II	Pareto envelope-based selection algorithm II		√		√	√	√						
121	PICEA-g	Preference-inspired coevolutionary algorithm with goals		√	√	√	√	√						
122	PM-MOEA	Pattern mining based multi-objective evolutionary algorithm		√		√			√	√				
123	POCEA	Paired offspring generation based constrained evolutionary algorithm		√		√			√	√				

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
124	PPS	Push and pull search algorithm		√	√	√				√				
125	PREA	Promising-region based EMO algorithm		√	√	√	√	√						
126	PSO	Particle swarm optimization	√			√			√	√				
127	RM-MEDA	Regularity model-based multiobjective estimation of distribution		√		√								
128	r-NSGA-II	r-dominance based NSGA-II		√		√	√	√						√
129	RPD-NSGA-II	Reference point dominance-based NSGA-II		√	√	√	√	√						
130	RPEA	Reference points-based evolutionary algorithm			√	√	√	√						
131	RSEA	Radial space division based evolutionary algorithm		√	√	√	√	√						
132	RVEA	Reference vector guided evolutionary algorithm		√	√	√	√	√		√				
133	RVEAa	RVEA embedded with the reference vector regeneration strategy			√	√	√	√						
134	RVEA-iGNG	RVEA based on improved growing neural gas		√	√	√	√	√						
135	S3-CMA-ES	Scalable small subpopulations based covariance matrix adaptation		√	√	√			√					
136	SA	Simulated annealing	√			√			√	√				
137	SACC-EAM-II	Surrogate-assisted cooperative co-evolutionary algorithm of Minamo	√			√					√			
138	SACOSO	Surrogate-assisted cooperative swarm optimization	√			√			√		√			
139	SADE-Sammon	Sammon mapping assisted differential evolution	√			√					√			
140	SAMSO	Multiswarm-assisted expensive optimization	√			√			√		√			
141	S-CDAS	Self-controlling dominance area of solutions			√	√	√	√						
142	SHADE	Success-history based adaptive differential evolution	√			√			√	√				
143	SIBEA	Simple indicator-based evolutionary algorithm		√		√	√	√						
144	SIBEA-kEMOSS	SIBEA with minimum objective subset of size k with minimum error			√	√	√	√						
145	SMEA	Self-organizing multiobjective evolutionary algorithm		√		√								
146	SMPSO	Speed-constrained multi-objective particle swarm optimization		√		√								
147	SMS-EGO	S metric selection based efficient global optimization		√		√					√			
148	SMS-EMOA	S metric selection based evolutionary multiobjective optimization		√		√	√	√						
149	SparseEA	Evolutionary algorithm for sparse multi-objective optimization problems		√		√	√		√	√			√	
150	SPEA2	Strength Pareto evolutionary algorithm 2		√		√	√	√						
151	SPEA2+SDE	SPEA2 with shift-based density estimation			√	√	√	√						
152	SPEA/R	Strength Pareto evolutionary algorithm based on reference direction		√	√	√	√	√						
153	SQP	Sequential quadratic programming	√			√			√	√				
154	SRA	Stochastic ranking algorithm			√	√	√	√						
155	t-DEA	theta-dominance based evolutionary algorithm		√	√	√	√	√						
156	TiGE-2	Tri-Goal Evolution Framework for CMaOPs			√	√	√	√		√				
157	ToP	Two-phase framework with NSGA-II		√		√				√				
158	TriMOEA-	Multi-modal MOEA using two-archive and		√		√						√		

No.	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
	TA&R	recombination strategies												
159	Two_Arch2	Two-archive algorithm 2		√	√	√	√	√						
160	VaEA	Vector angle based evolutionary algorithm		√	√	√	√	√						
161	WOF	Weighted optimization framework		√		√			√					
162	WV-MOEA-P	Weight vector based multi-objective optimization algorithm with preference		√		√								√

VI. List of Problems

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
1	BT1	Benchmark MOP with bias feature		√		√			√					
2	BT2	Benchmark MOP with bias feature		√		√			√					
3	BT3	Benchmark MOP with bias feature		√		√			√					
4	BT4	Benchmark MOP with bias feature		√		√			√					
5	BT5	Benchmark MOP with bias feature		√		√			√					
6	BT6	Benchmark MOP with bias feature		√		√			√					
7	BT7	Benchmark MOP with bias feature		√		√			√					
8	BT8	Benchmark MOP with bias feature		√		√			√					
9	BT9	Benchmark MOP with bias feature		√		√			√					
10	CEC2008_F1	Shifted sphere function	√			√			√		√			
11	CEC2008_F2	Shifted Schwefel's function	√			√			√		√			
12	CEC2008_F3	Shifted Rosenbrock's function	√			√			√		√			
13	CEC2008_F4	Shifted Rastrign's function	√			√			√		√			
14	CEC2008_F5	Shifted Griewank's function	√			√			√		√			
15	CEC2008_F6	Shifted Ackley's function	√			√			√		√			
16	CEC2008_F7	FastFractal 'DoubleDip' function	√			√			√		√			
17	CEC2010_F1	CEC'2010 constrained optimization benchmark problem	√			√				√				
18	CEC2010_F2	CEC'2010 constrained optimization benchmark problem	√			√				√				
19	CEC2010_F3	CEC'2010 constrained optimization benchmark problem	√			√				√				
20	CEC2010_F4	CEC'2010 constrained optimization benchmark problem	√			√				√				
21	CEC2010_F5	CEC'2010 constrained optimization benchmark problem	√			√				√				
22	CEC2010_F6	CEC'2010 constrained optimization benchmark problem	√			√				√				
23	CEC2010_F7	CEC'2010 constrained optimization benchmark problem	√			√				√				
24	CEC2010_F8	CEC'2010 constrained optimization benchmark problem	√			√				√				
25	CEC2010_F9	CEC'2010 constrained optimization benchmark problem	√			√				√				
26	CEC2010_F10	CEC'2010 constrained optimization benchmark problem	√			√				√				
27	CEC2010_F11	CEC'2010 constrained optimization benchmark problem	√			√				√				
28	CEC2010_F12	CEC'2010 constrained optimization benchmark problem	√			√				√				
29	CEC2010_F13	CEC'2010 constrained optimization benchmark problem	√			√				√				
30	CEC2010_F14	CEC'2010 constrained optimization benchmark problem	√			√				√				
31	CEC2010_F15	CEC'2010 constrained optimization benchmark problem	√			√				√				
32	CEC2010_F16	CEC'2010 constrained optimization benchmark problem	√			√				√				
33	CEC2010_F17	CEC'2010 constrained optimization benchmark problem	√			√				√				

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
34	CEC2010_F18	CEC'2010 constrained optimization benchmark problem	√			√				√				
35	CEC2013_F1	Shifted elliptic function	√			√			√					
36	CEC2013_F2	Shifted Rastrigin's function	√			√			√					
37	CEC2013_F3	Shifted Ackley's function	√			√			√					
38	CEC2013_F4	7-nonseparable, 1-separable shifted and rotated elliptic function	√			√			√					
39	CEC2013_F5	7-nonseparable, 1-separable shifted and rotated Rastrigin's function	√			√			√					
40	CEC2013_F6	7-nonseparable, 1-separable shifted and rotated Ackley's function	√			√			√					
41	CEC2013_F7	7-nonseparable, 1-separable shifted and rotated Schwefel's function	√			√			√					
42	CEC2013_F8	20-nonseparable shifted and rotated elliptic function	√			√			√					
43	CEC2013_F9	20-nonseparable shifted and rotated Rastrigin's function	√			√			√					
44	CEC2013_F10	20-nonseparable shifted and rotated Rastrigin's function	√			√			√					
45	CEC2013_F11	20-nonseparable shifted and rotated Schwefel's function	√			√			√					
46	CEC2013_F12	Shifted Rosenbrock's function	√			√			√					
47	CEC2013_F13	Shifted Schwefel's function with conforming overlapping subcomponents	√			√			√					
48	CEC2013_F14	Shifted Schwefel's function with conflicting overlapping subcomponents	√			√			√					
49	CEC2013_F15	Shifted Schwefel's function	√			√			√					
50	CEC2017_F1	CEC'2017 constrained optimization benchmark problem	√			√				√				
51	CEC2017_F2	CEC'2017 constrained optimization benchmark problem	√			√				√				
52	CEC2017_F3	CEC'2017 constrained optimization benchmark problem	√			√				√				
53	CEC2017_F4	CEC'2017 constrained optimization benchmark problem	√			√				√				
54	CEC2017_F5	CEC'2017 constrained optimization benchmark problem	√			√				√				
55	CEC2017_F6	CEC'2017 constrained optimization benchmark problem	√			√				√				
56	CEC2017_F7	CEC'2017 constrained optimization benchmark problem	√			√				√				
57	CEC2017_F8	CEC'2017 constrained optimization benchmark problem	√			√				√				
58	CEC2017_F9	CEC'2017 constrained optimization benchmark problem	√			√				√				
59	CEC2017_F10	CEC'2017 constrained optimization benchmark problem	√			√				√				
60	CEC2017_F11	CEC'2017 constrained optimization benchmark problem	√			√				√				
61	CEC2017_F12	CEC'2017 constrained optimization benchmark problem	√			√				√				
62	CEC2017_F13	CEC'2017 constrained optimization benchmark problem	√			√				√				
63	CEC2017_F14	CEC'2017 constrained optimization benchmark problem	√			√				√				
64	CEC2017_F15	CEC'2017 constrained optimization benchmark problem	√			√				√				
65	CEC2017_F16	CEC'2017 constrained optimization benchmark problem	√			√				√				
66	CEC2017_F17	CEC'2017 constrained optimization benchmark problem	√			√				√				
67	CEC2017_F18	CEC'2017 constrained optimization benchmark problem	√			√				√				
68	CEC2017_F19	CEC'2017 constrained optimization benchmark problem	√			√				√				
69	CEC2017_F20	CEC'2017 constrained optimization benchmark problem	√			√				√				

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
70	CEC2017_F21	CEC'2017 constrained optimization benchmark problem	√			√				√				
71	CEC2017_F22	CEC'2017 constrained optimization benchmark problem	√			√				√				
72	CEC2017_F23	CEC'2017 constrained optimization benchmark problem	√			√				√				
73	CEC2017_F24	CEC'2017 constrained optimization benchmark problem	√			√				√				
74	CEC2017_F25	CEC'2017 constrained optimization benchmark problem	√			√				√				
75	CEC2017_F26	CEC'2017 constrained optimization benchmark problem	√			√				√				
76	CEC2017_F27	CEC'2017 constrained optimization benchmark problem	√			√				√				
77	CEC2017_F28	CEC'2017 constrained optimization benchmark problem	√			√				√				
78	CEC2020_F1	Bent cigar function	√			√								
79	CEC2020_F2	Shifted and rotated Schwefel's function	√			√								
80	CEC2020_F3	Shifted and rotated Lunacek bi-Rastrigin function	√			√								
81	CEC2020_F4	Expanded Rosenbrock's plus Griewangk's function	√			√								
82	CEC2020_F5	Hybrid function 1	√			√								
83	CEC2020_F6	Hybrid function 2	√			√								
84	CEC2020_F7	Hybrid function 3	√			√								
85	CEC2020_F8	Composition function 1	√			√								
86	CEC2020_F9	Composition function 2	√			√								
87	CEC2020_F10	Composition function 3	√			√								
88	CF1	Constrained benchmark MOP		√		√			√	√				
89	CF2	Constrained benchmark MOP		√		√			√	√				
90	CF3	Constrained benchmark MOP		√		√			√	√				
91	CF4	Constrained benchmark MOP		√		√			√	√				
92	CF5	Constrained benchmark MOP		√		√			√	√				
93	CF6	Constrained benchmark MOP		√		√			√	√				
94	CF7	Constrained benchmark MOP		√		√			√	√				
95	CF8	Constrained benchmark MOP		√		√			√	√				
96	CF9	Constrained benchmark MOP		√		√			√	√				
97	CF10	Constrained benchmark MOP		√		√			√	√				
98	DAS-CMOP1	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
99	DAS-CMOP2	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
100	DAS-CMOP3	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
101	DAS-CMOP4	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
102	DAS-CMOP5	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
103	DAS-CMOP6	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
104	DAS-CMOP7	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
105	DAS-CMOP8	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				
106	DAS-CMOP9	Difficulty-adjustable and scalable constrained benchmark MOP		√		√			√	√				

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
107	DOC1	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
108	DOC2	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
109	DOC3	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
110	DOC4	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
111	DOC5	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
112	DOC6	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
113	DOC7	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
114	DOC8	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
115	DOC9	Benchmark MOP with constraints in decision and objective spaces		√		√				√				
116	DTLZ1	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√		√			
117	DTLZ2	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√		√			
118	DTLZ3	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√		√			
119	DTLZ4	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√		√			
120	DTLZ5	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√		√			
121	DTLZ6	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√		√			
122	DTLZ7	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√		√			
123	DTLZ8	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√	√	√			
124	DTLZ9	Benchmark MOP proposed by Deb, Thiele, Laumanns, and Zitzler		√	√	√			√	√	√			
125	CDTLZ2	Convex DTLZ2		√	√	√			√		√			
126	IDTLZ1	Inverted DTLZ1		√	√	√			√		√			
127	IDTLZ2	Inverted DTLZ2		√	√	√			√		√			
128	SDTLZ1	Scaled DTLZ1		√	√	√			√		√			
129	SDTLZ2	Scaled DTLZ2		√	√	√			√		√			
130	C1-DTLZ1	Constrained DTLZ1		√	√	√			√	√	√			
131	C1-DTLZ3	Constrained DTLZ3		√	√	√			√	√	√			
132	C2-DTLZ2	Constrained DTLZ2		√	√	√			√	√	√			
133	C3-DTLZ4	Constrained DTLZ4		√	√	√			√	√	√			
134	DC1-DTLZ1	DTLZ1 with constrains in decision space		√	√	√			√	√	√			
135	DC1-DTLZ3	DTLZ3 with constrains in decision space		√	√	√			√	√	√			
136	DC2-DTLZ1	DTLZ1 with constrains in decision space		√	√	√			√	√	√			
137	DC2-DTLZ3	DTLZ3 with constrains in decision space		√	√	√			√	√	√			
138	DC3-DTLZ1	DTLZ1 with constrains in decision space		√	√	√			√	√	√			
139	DC3-DTLZ3	DTLZ3 with constrains in decision space		√	√	√			√	√	√			
140	FCP1	Benchmark constrained MOP proposed by Yuan		√		√				√				
141	FCP2	Benchmark constrained MOP proposed by Yuan		√		√				√				
142	FCP3	Benchmark constrained MOP proposed by Yuan		√		√				√				
143	FCP4	Benchmark constrained MOP proposed by Yuan		√		√				√				

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
144	FCP5	Benchmark constrained MOP proposed by Yuan		√		√				√				
145	IMMOEA_F1	Benchmark MOP for testing IM-MOEA		√		√			√					
146	IMMOEA_F2	Benchmark MOP for testing IM-MOEA		√		√			√					
147	IMMOEA_F3	Benchmark MOP for testing IM-MOEA		√		√			√					
148	IMMOEA_F4	Benchmark MOP for testing IM-MOEA		√		√			√					
149	IMMOEA_F5	Benchmark MOP for testing IM-MOEA		√		√			√					
150	IMMOEA_F6	Benchmark MOP for testing IM-MOEA		√		√			√					
151	IMMOEA_F7	Benchmark MOP for testing IM-MOEA		√		√			√					
152	IMMOEA_F8	Benchmark MOP for testing IM-MOEA		√		√			√					
153	IMMOEA_F9	Benchmark MOP for testing IM-MOEA		√		√			√					
154	IMMOEA_F10	Benchmark MOP for testing IM-MOEA		√		√			√					
155	IMOP1	Benchmark MOP with irregular Pareto front		√		√					√			
156	IMOP2	Benchmark MOP with irregular Pareto front		√		√					√			
157	IMOP3	Benchmark MOP with irregular Pareto front		√		√					√			
158	IMOP4	Benchmark MOP with irregular Pareto front		√		√					√			
159	IMOP5	Benchmark MOP with irregular Pareto front		√		√					√			
160	IMOP6	Benchmark MOP with irregular Pareto front		√		√					√			
161	IMOP7	Benchmark MOP with irregular Pareto front		√		√					√			
162	IMOP8	Benchmark MOP with irregular Pareto front		√		√					√			
163	KP	The knapsack problem	√				√		√	√				
164	LIR-CMOP1	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
165	LIR-CMOP2	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
166	LIR-CMOP3	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
167	LIR-CMOP4	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
168	LIR-CMOP5	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
169	LIR-CMOP6	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
170	LIR-CMOP7	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
171	LIR-CMOP8	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
172	LIR-CMOP9	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
173	LIR-CMOP10	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
174	LIR-CMOP11	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
175	LIR-CMOP12	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
176	LIR-CMOP13	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
177	LIR-CMOP14	Constrained benchmark MOP with large infeasible regions		√		√			√	√				
178	LSMOP1	Large-scale benchmark MOP		√	√	√			√					
179	LSMOP2	Large-scale benchmark MOP		√	√	√			√					
180	LSMOP3	Large-scale benchmark MOP		√	√	√			√					

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
181	LSMOP4	Large-scale benchmark MOP		√	√	√			√					
182	LSMOP5	Large-scale benchmark MOP		√	√	√			√					
183	LSMOP6	Large-scale benchmark MOP		√	√	√			√					
184	LSMOP7	Large-scale benchmark MOP		√	√	√			√					
185	LSMOP8	Large-scale benchmark MOP		√	√	√			√					
186	LSMOP9	Large-scale benchmark MOP		√	√	√			√					
187	MaF1	Inverted DTLZ1		√	√	√			√					
188	MaF2	DTLZ2BZ		√	√	√			√					
189	MaF3	Convex DTLZ3		√	√	√			√					
190	MaF4	Inverted and scaled DTLZ3		√	√	√			√					
191	MaF5	Scaled DTLZ4		√	√	√			√					
192	MaF6	DTLZ5IM		√	√	√			√					
193	MaF7	DTLZ7		√	√	√			√					
194	MaF8	MP-DMP		√	√	√								
195	MaF9	ML-DMP		√	√	√								
196	MaF10	WFG1		√	√	√			√					
197	MaF11	WFG2		√	√	√			√					
198	MaF12	WFG9		√	√	√			√					
199	MaF13	P7		√	√	√			√					
200	MaF14	LSMOP3		√	√	√			√					
201	MaF15	Inverted LSMOP8		√	√	√			√					
202	MLDMP	The multi-line distance minimization problem		√	√	√								
203	MMF1	Multi-modal multi-objective test function		√		√						√		
204	MMF2	Multi-modal multi-objective test function		√		√						√		
205	MMF3	Multi-modal multi-objective test function		√		√						√		
206	MMF4	Multi-modal multi-objective test function		√		√						√		
207	MMF5	Multi-modal multi-objective test function		√		√						√		
208	MMF6	Multi-modal multi-objective test function		√		√						√		
209	MMF7	Multi-modal multi-objective test function		√		√						√		
210	MMF8	Multi-modal multi-objective test function		√		√						√		
211	MMMOP1	Multi-modal multi-objective optimization problem		√	√	√						√		
212	MMMOP2	Multi-modal multi-objective optimization problem		√	√	√						√		
213	MMMOP3	Multi-modal multi-objective optimization problem		√	√	√						√		
214	MMMOP4	Multi-modal multi-objective optimization problem		√	√	√						√		
215	MMMOP5	Multi-modal multi-objective optimization problem		√	√	√						√		
216	MMMOP6	Multi-modal multi-objective optimization problem		√	√	√						√		
217	MOEADDE_F1	Benchmark MOP for testing MOEA/D-DE		√		√			√					

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
218	MOEADDE_F2	Benchmark MOP for testing MOEA/D-DE		√		√			√					
219	MOEADDE_F3	Benchmark MOP for testing MOEA/D-DE		√		√			√					
220	MOEADDE_F4	Benchmark MOP for testing MOEA/D-DE		√		√			√					
221	MOEADDE_F5	Benchmark MOP for testing MOEA/D-DE		√		√			√					
222	MOEADDE_F6	Benchmark MOP for testing MOEA/D-DE		√		√			√					
223	MOEADDE_F7	Benchmark MOP for testing MOEA/D-DE		√		√			√					
224	MOEADDE_F8	Benchmark MOP for testing MOEA/D-DE		√		√			√					
225	MOEADDE_F9	Benchmark MOP for testing MOEA/D-DE		√		√			√					
226	MOEADM2M_F1	Benchmark MOP for testing MOEA/D-M2M		√		√			√					
227	MOEADM2M_F2	Benchmark MOP for testing MOEA/D-M2M		√		√			√					
228	MOEADM2M_F3	Benchmark MOP for testing MOEA/D-M2M		√		√			√					
229	MOEADM2M_F4	Benchmark MOP for testing MOEA/D-M2M		√		√			√					
230	MOEADM2M_F5	Benchmark MOP for testing MOEA/D-M2M		√		√			√					
231	MOEADM2M_F6	Benchmark MOP for testing MOEA/D-M2M		√		√			√					
232	MOEADM2M_F7	Benchmark MOP for testing MOEA/D-M2M		√		√			√					
233	MOKP	The multi-objective knapsack problem		√	√		√		√					
234	MONRP	The multi-objective next release problem		√			√		√					
235	MOTSP	The multi-objective traveling salesman problem		√	√			√	√					
236	MPDMP	The multi-point distance minimization problem		√	√	√								
237	mQAP	The multi-objective quadratic assignment problem		√	√			√	√					
238	MW1	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
239	MW2	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
240	MW3	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
241	MW4	Constrained benchmark MOP proposed by Ma and Wang		√	√	√			√	√				
242	MW5	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
243	MW6	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
244	MW7	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
245	MW8	Constrained benchmark MOP proposed by Ma and Wang		√	√	√			√	√				
246	MW9	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
247	MW10	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
248	MW11	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
249	MW12	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
250	MW13	Constrained benchmark MOP proposed by Ma and Wang		√		√			√	√				
251	MW14	Constrained benchmark MOP proposed by Ma and Wang		√	√	√			√	√				
252	RMMEDA_F1	Benchmark MOP for testing RM-MEDA		√		√			√					
253	RMMEDA_F2	Benchmark MOP for testing RM-MEDA		√		√			√					
254	RMMEDA_F3	Benchmark MOP for testing RM-MEDA		√		√			√					

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
255	RMMEDA_F4	Benchmark MOP for testing RM-MEDA		√		√			√					
256	RMMEDA_F5	Benchmark MOP for testing RM-MEDA		√		√			√					
257	RMMEDA_F6	Benchmark MOP for testing RM-MEDA		√		√			√					
258	RMMEDA_F7	Benchmark MOP for testing RM-MEDA		√		√			√					
259	RMMEDA_F8	Benchmark MOP for testing RM-MEDA		√		√			√					
260	RMMEDA_F9	Benchmark MOP for testing RM-MEDA		√		√			√					
261	RMMEDA_F10	Benchmark MOP for testing RM-MEDA		√		√			√					
262	Sparse_CD	The community detection problem		√			√		√		√		√	
263	Sparse_CN	The critical node detection problem		√			√		√		√		√	
264	Sparse_FS	The feature selection problem		√			√		√		√		√	
265	Sparse_IS	The instance selection problem		√			√		√		√		√	
266	Sparse_KP	The sparse multi-objective knapsack problem		√	√		√		√					
267	Sparse_NN	The neural network training problem		√		√			√		√		√	
268	Sparse_PM	The pattern mining problem		√			√		√		√		√	
269	Sparse_PO	The portfolio optimization problem		√		√			√		√		√	
270	Sparse_SR	The sparse signal reconstruction problem		√		√			√		√		√	
271	SMMOP1	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
272	SMMOP2	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
273	SMMOP3	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
274	SMMOP4	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
275	SMMOP5	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
276	SMMOP6	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
277	SMMOP7	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
278	SMMOP8	Sparse multi-modal multi-objective optimization problem		√	√	√			√			√	√	
279	SMOP1	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
280	SMOP2	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
281	SMOP3	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
282	SMOP4	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
283	SMOP5	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
284	SMOP6	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
285	SMOP7	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
286	SMOP8	Benchmark MOP with sparse Pareto optimal solutions		√	√	√			√		√		√	
287	SOP_F1	Sphere function	√			√					√			
288	SOP_F2	Schwefel's function 2.22	√			√					√			
289	SOP_F3	Schwefel's function 1.2	√			√					√			
290	SOP_F4	Schwefel's function 2.21	√			√					√			
291	SOP_F5	Generalized Rosenbrock's function	√			√					√			

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
292	SOP_F6	Step function	√			√					√			
293	SOP_F7	Quartic function with noise	√			√					√			
294	SOP_F8	Generalized Schwefel's function 2.26	√			√					√			
295	SOP_F9	Generalized Rastrigin's function	√			√					√			
296	SOP_F10	Ackley's function	√			√					√			
297	SOP_F11	Generalized Griewank's function	√			√					√			
298	SOP_F12	Generalized penalized function	√			√					√			
299	SOP_F13	Generalized penalized function	√			√					√			
300	SOP_F14	Shekel's foxholes function	√			√					√			
301	SOP_F15	Kowalik's function	√			√					√			
302	SOP_F16	Six-hump camel-back function	√			√					√			
303	SOP_F17	Branin function	√			√					√			
304	SOP_F18	Goldstein-price function	√			√					√			
305	SOP_F19	Hartman's family	√			√					√			
306	SOP_F20	Hartman's family	√			√					√			
307	SOP_F21	Shekel's family	√			√					√			
308	SOP_F22	Shekel's family	√			√					√			
309	SOP_F23	Shekel's family	√			√					√			
310	TREE1	The time-varying ratio error estimation problem		√		√			√	√	√			
311	TREE2	The time-varying ratio error estimation problem		√		√			√	√	√			
312	TREE3	The time-varying ratio error estimation problem		√		√			√	√	√			
313	TREE4	The time-varying ratio error estimation problem		√		√			√	√	√			
314	TREE5	The time-varying ratio error estimation problem		√		√			√	√	√			
315	TREE6	The time-varying ratio error estimation problem		√		√			√	√	√			
316	TSP	The traveling salesman problem	√					√	√					
317	UF1	Unconstrained benchmark MOP		√		√			√					
318	UF2	Unconstrained benchmark MOP		√		√			√					
319	UF3	Unconstrained benchmark MOP		√		√			√					
320	UF4	Unconstrained benchmark MOP		√		√			√					
321	UF5	Unconstrained benchmark MOP		√		√			√					
322	UF6	Unconstrained benchmark MOP		√		√			√					
323	UF7	Unconstrained benchmark MOP		√		√			√					
324	UF8	Unconstrained benchmark MOP		√		√			√					
325	UF9	Unconstrained benchmark MOP		√		√			√					
326	UF10	Unconstrained benchmark MOP		√		√			√					
327	VNT1	Benchmark MOP proposed by Viennet		√		√								
328	VNT2	Benchmark MOP proposed by Viennet		√		√								

	Abbreviation	Full name	single	multi	many	real	binary	permutation	large	constrained	expensive	multimodal	sparse	preference
329	VNT3	Benchmark MOP proposed by Viennet		√		√								
330	VNT4	Benchmark MOP proposed by Viennet		√		√				√				
331	WFG1	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
332	WFG2	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
333	WFG3	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
334	WFG4	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
335	WFG5	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
336	WFG6	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
337	WFG7	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
338	WFG8	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
339	WFG9	Benchmark MOP proposed by Walking Fish Group		√	√	√			√		√			
340	ZDT1	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√			√		√			
341	ZDT2	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√			√		√			
342	ZDT3	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√			√		√			
343	ZDT4	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√			√		√			
344	ZDT5	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√			√		√		√			
345	ZDT6	Benchmark MOP proposed by Zitzler, Deb, and Thiele		√		√			√		√			