Neurocomputing 518 (2023) 190–205

Contents lists available at ScienceDirect

Neurocomputing

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

A practical tutorial on solving optimization problems via PlatEMO

Ye Tian [a,b], Weijian Zhu [c], Xingyi Zhang [a,d,*], Yaochu Jin [e]

[a] Information Materials and Intelligent Sensing Laboratory of Anhui Province, Anhui University, Hefei, China
[b] Institutes of Physical Science and Information Technology, Anhui University, Hefei, China
[c] School of Computer Science and Technology, Anhui University, Hefei, China
[d] School of Artificial Intelligence, Anhui University, China
[e] Faculty of Technology, Bielefeld University, Bielefeld, Germany

ARTICLE INFO

Article history:
Received 6 September 2022
Revised 24 October 2022
Accepted 30 October 2022
Available online 10 November 2022
Communicated by Zidong Wang

Keywords:
Optimization
Metaheuristics
Evolutionary computation
Swarm intelligence
Problem definition
PlatEMO

ABSTRACT

PlatEMO is an open-source platform for solving complex optimization problems, which provides a variety of metaheuristics including evolutionary algorithms, swarm intelligence algorithms, multi-objective optimization algorithms, surrogate-assisted optimization algorithms, and many others. Due to the problem-independent nature of most metaheuristics, they are versatile for solving problems with various difficulties such as multimodal landscapes, discrete search spaces, multiple objectives, strict constraints, and expensive evaluations, regardless of the fields the problems belong to. Since PlatEMO was published in 2017, it has been used by many researchers from both academia and industry in the computational intelligence community. However, the basic terms and concepts about optimization may confuse practitioners and junior researchers new to metaheuristics. Hence, this paper presents a practical introduction to the use of PlatEMO 4.0, focusing on the procedures of defining problems, selecting suitable metaheuristics, and collecting results. Note, however, that a description of the technical details of metaheuristics is beyond the scope of this paper and interested readers may refer to the cited references.

1. Introduction

Optimization problems widely exist as key tasks in many scientific and engineering fields, which become more and more challenging in the era of big data [1–3]. After decades of development, metaheuristics have shown effectiveness in solving various optimization problems, and are getting increasingly important with the problem complexity increases [4]. As an artificial intelligence technology, metaheuristics are popular for their high versatility in handling complex optimization problems. On the one hand, metaheuristics search for optimal solutions in a black-box manner, where a single metaheuristic can be applied to various fields without using problem-dependent information (e.g., gradients or datasets) [5]. On the other hand, metaheuristics make few assumptions about the problem being solved, where discrete variables, multiple objectives, and strict constraints can be directly handled without conversion or aggregation [6]. In short, metaheuristics are plug and play optimizers, in which the elimination of cumbersome configuration facilitates the optimization of complex problems for non-expert users.

Nevertheless, the utilization of metaheuristics is not straightforward, as there exist too many metaheuristics rather than too few [7]. So far, a large number of metaphor-inspired metaheuristics have attracted criticism for hiding their lack of novelty behind an elaborate metaphor [8]. While the performance of metaheuristics was mainly studied on benchmarks in the papers, it is difficult for users to identify the most effective one to solve specific applications [9]. More seriously, each metaheuristic has its own application scope, the misuse of which may not only lead to unsatisfactory performance but also raise errors. If a decomposition based multi-objective metaheuristic [10] is used to solve a single-objective optimization problem, the program will be stuck in an infinite loop. If a principal component analysis based metaheuristic [11] is used to solve a problem with 10000 variables, the program may get out of memory. If a large-scale metaheuristic [12] is used to solve an expensive problem, the program may run for several months before termination.

In 2017, we developed a platform for solving complex optimization problems, call evolutionary multi-objective optimization platform (PlatEMO) [13]. Although PlatEMO was named by *evolutionary computation* and *multi-objective optimization*, after more than forty updates, it currently contains various advanced metaheuristics and other optimizers for solving single-objective, multi-objective, and other types of optimization problems

* Corresponding author.
E-mail address: xyzhanghust@gmail.com (X. Zhang).

https://doi.org/10.1016/j.neucom.2022.10.075
0925-2312/

efficiently [14]. To reveal the application scopes of metaheuristics, PlatEMO tags each metaheuristic with several labels indicating the types of optimization problems, which provides a clear guidance for selecting suitable metaheuristics. To solve an optimization problem via PlatEMO, users should define the problem and identify its labels, then select one metaheuristic tagged with the same labels, and collect the optimization results in several formats (i.e., matrices, figures, or tables). These operations are not easy for practitioners unfamiliar with programming, and they may be confused by the basic concepts about optimization from the very beginning, such as the terminologies illustrated in Fig. 1.

Therefore, this paper presents the detailed procedures of the above operations, together with an introduction to some basic concepts about optimization. We do not introduce the technical details of metaheuristics in this paper, where interested readers may refer to some surveys on classical metaheuristics [7,15] and the state-of-the-art [4,16]. By contrast, we aim to enable beginners to use PlatEMO at a low cost. To do so, Section 2 introduces the basic functions of PlatEMO, Section 3 presents the procedures of defining optimization problems, Section 4 presents the procedures of selecting suitable metaheuristics, Section 5 presents the procedures of collecting optimization results, and Section 6 draws some conclusions. This paper is written according to many questions raised by users, which are common sense to professional researchers but likely to confuse users out of the computational intelligence community.

2. Basic functions of PlatEMO

PlatEMO is an open-source platform developed based on MATLAB, which consists of source codes (`.m` files) and datasets (`.mat` files) [17]. The newest version PlatEMO 4.0 can be downloaded from https://github.com/BIMK/PlatEMO, where all the descriptions in the rest of this paper are applicable to this version.

To use PlatEMO, users should make sure that the folder and subfolders of PlatEMO have been added to the search path of MATLAB, then call the main function like.

```
platemo();
```

Then a graphical user interface (GUI) will be displayed, which contains three modules with different functions. As plotted in Fig. 2, the test module is used to visually investigate the performance of a metaheuristic on an existing problem with the following steps:

Step (1) Select multiple labels to filter out the desired metaheuristics and problems. Meanings of the labels are introduced in Section 4.
Step (2) Select one metaheuristic from the list of desired metaheuristics.
Step (3) Select one problem from the list of desired problems. The way of creating a problem is introduced in Section 3.3.
Step (4) Set the parameters of the selected metaheuristic and problem. A parameter is set to its default value if the textbox is empty.
Step (5) Start the optimization process; pause, stop, and back off the current execution; save the current optimization result to files. Details of the saved files are introduced in Section 5.1.
Step (6) Select the data (e.g., decision variables or objective values) of the current result and display it on the axis; save, rotate, pan, and zoom the axis.
Step (7) Select a historical result and display it on the axis; calculate a performance metric value of the result. Details of the performance metrics are introduced in Section 5.3.

As plotted in Fig. 3, the application module is used to define a new problem and solve it via suitable metaheuristics with the following steps:

Step (1) Define the details of a problem. The way of defining problems is introduced in Section 3.2.
Step (2) Save, load, and validate the problem; select a predefined problem template.
Step (3) Select one metaheuristic from the list of desired metaheuristics. The labels are automatically determined according to the problem definition.
Step (4) Set the parameters of the selected metaheuristic. A parameter is set to its default value if the textbox is empty.
Step (5) Start the optimization process; pause, stop, and back off the current execution; save the current optimization result to files. Details of the saved files are introduced in Section 5.1.
Step (6) Select the data (e.g., decision variables or objective values) of the current result and display it on the axis; save, rotate, pan, and zoom the axis.

As plotted in Fig. 4, the experiment module is used to statistically investigate the performance of multiple metaheuristics on multiple existing problems with the following steps:

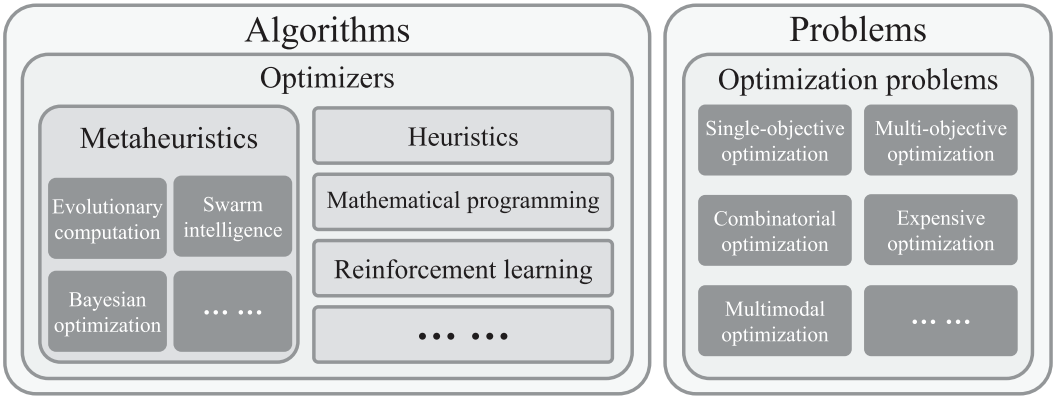

Fig. 1. Relevant terminologies in optimization and their relationships.

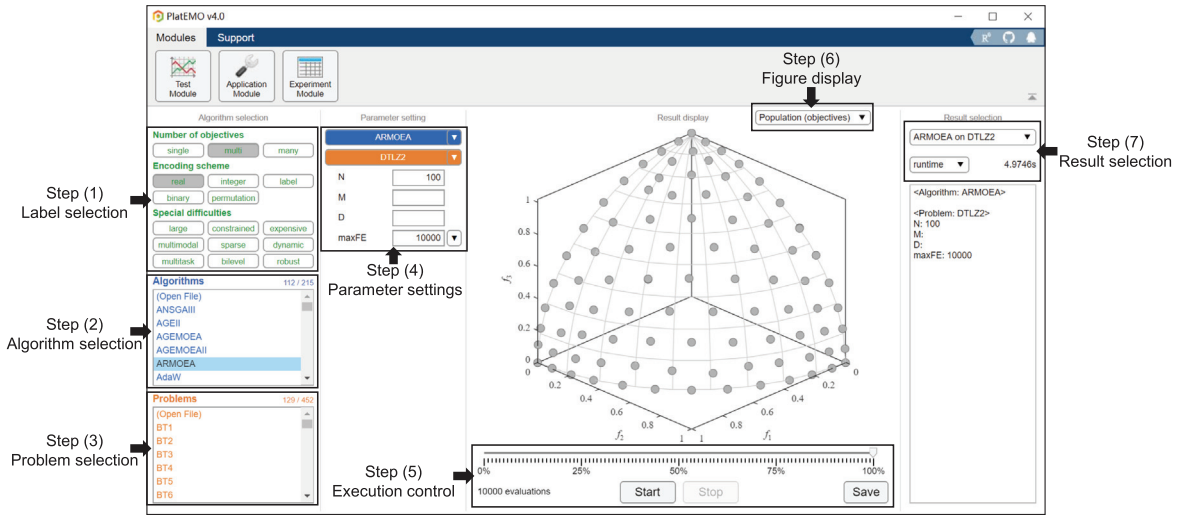

Fig. 2. Interface of the test module of PlatEMO.

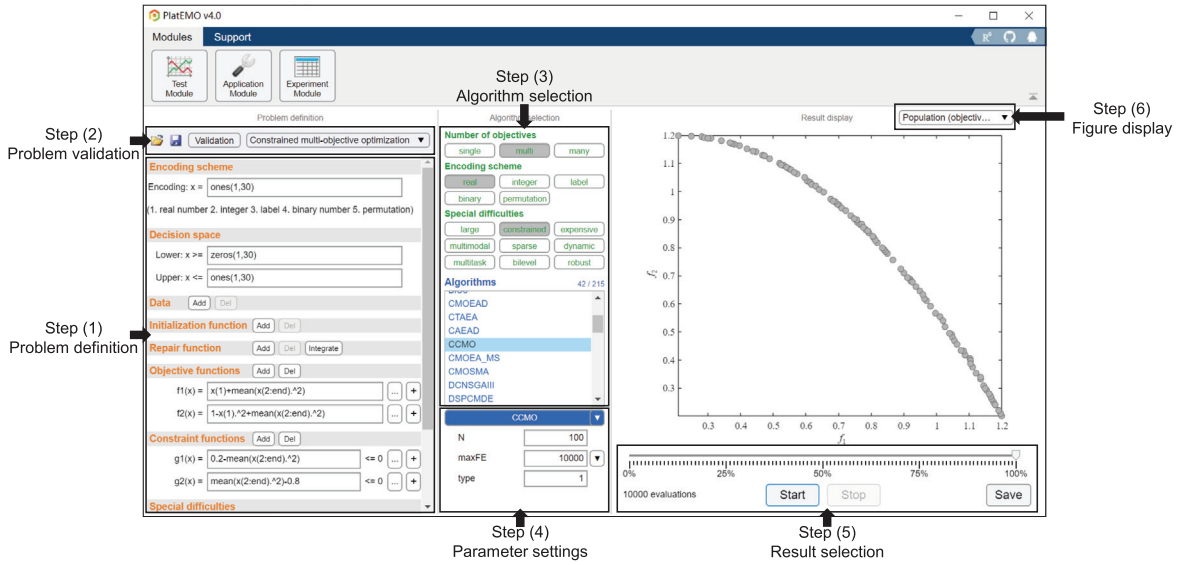

Fig. 3. Interface of the application module of PlatEMO.

Step (1) Select multiple labels to filter out the desired metaheuristics and problems. Meanings of the labels are introduced in Section 4.

Step (2) Select multiple metaheuristics from the list of desired metaheuristics.

Step (3) Select multiple problems from the list of desired problems. The way of creating problems is introduced in Section 3.3.

Step (4) Set the number of runs, number of solution sets saved in each run, and the path for saving optimization results. The result of each run is saved to a file, whose details are introduced in Section 5.2. If the file exists, it will be loaded and one run is skipped.

Step (5) Set the parameters of the selected metaheuristics and problems. A parameter is set to its default value if the textbox is empty.

Step (6) Start the optimization process; pause and stop the current execution; use or not use all CPUs to execute multiple runs in parallel.

Step (7) Select a performance metric and show the metric values in the table; show or hide statistical results; save the table to an Excel, TeX, TXT, or MAT file; display the results of multiple selected cells in multiple axes.

On the other hand, users can call the main function with inputs to use the command mode of PlatEMO, where no GUI will be displayed. The inputs consist of multiple pairs of variable names and values, where all the acceptable variables are summarized in Table 1. Users need not specify all the variables as each of them has a default value. Details of these variables are described as follows.

- 'algorithm' denotes the metaheuristic to be executed, whose value should be a function handle or a cell. For example,

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

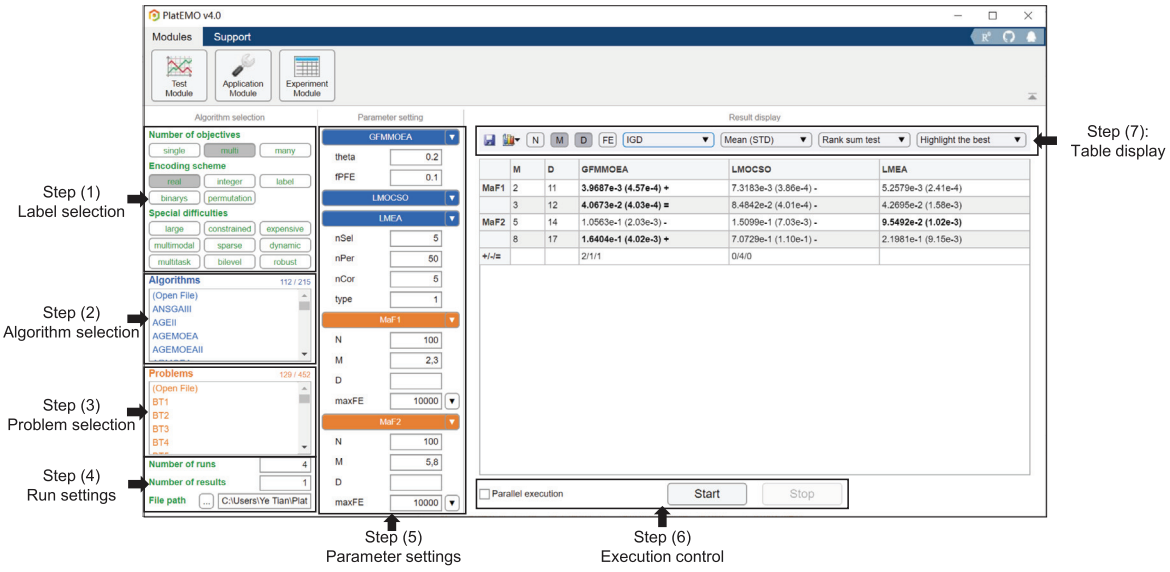

Fig. 4. Interface of the experiment module of PlatEMO.

Table 1
Details of the acceptable variable names in the command mode of PlatEMO.

Name	Data type	Default value	Description
'algorithm'	Function handle or cell	N/A	A metaheuristic and its parameters
'problem'	Function handle or cell	N/A	A problem and its parameters
'N'	Positive integer	100	Solution set size of the metaheuristic
'M'	Positive integer	N/A	Number of objectives of the problem
'D'	Positive integer	N/A	Number of decision variables of the problem
'maxFE'	Positive integer	10000	Maximum number of function evaluations before termination
'maxRuntime'	Positive number	inf	Maximum runtime (in second) before termination
'save'	Integer	-10	Number of saved solution sets
'outputFcn'	Function handle	@DefaultOutput	Function called before each iteration of the metaheuristic
'objFcn'	Function handle or cell	{}	Objective functions
'encoding'	Row vector	1	Encoding scheme of the problem (1 = real 2 = integer 3 = label 4 = binary 5 = permutation)
'lower'	Row vector	0	Lower bounds of varaibles
'upper'	Row vector	1	Upper bounds of varaibles
'conFcn'	Function handle or cell	{}	Constraint functions
'decFcn'	Function handle	{}	Function for repairing an invalid solution
'evalFcn'	Function handle	{}	Function for evaluating a solution
'initFcn'	Function handle	{}	Function for initializing a solution set
'objGradFcn'	Function handle or cell	{}	Functions for calculating the gradients of objectives
'conGradFcn'	Function handle or cell	{}	Functions for calculating the gradients of constraints
'data'	Cell	{}	Data of the problem

uses the genetic algorithm [18] with the default settings to solve the default problem, and

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

uses the genetic algorithm with the specified settings to solve the default problem.

- 'problem' denotes the problem to be solved, whose value should be a function handle or a cell. For example,

```
platemo('problem',@DTLZ1);
```

uses the default metaheuristic to solve DTLZ1 [19] with the default settings, and

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

uses NSGA-II [20] to solve WFG1 [21] with the specified settings. The way of creating problems is introduced in Section 3.3.

- 'N' denotes the solution set (i.e., population) size of the metaheuristic, where the solution set is the optimization result. The solution set size of some metaheuristics cannot be arbitrarily set. For example,

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

sets the solution set size of the genetic algorithm to 50.

- 'M' denotes the number of objectives of the problem. The number of objectives of some problems cannot be arbitrarily set. For example,

```
platemo('problem',@DTLZ1,'M',10);
```

sets the number of objectives of DTLZ1 to 10.

- 'D' denotes the number of decision variables of the problem. The number of decision variables of some problems cannot be arbitrarily set. For example,

```
platemo('problem',@MaF1,'D',50);
```

sets the number of decision variables of MaF1 [22] to 50.

- 'maxFE' and 'maxRuntime' denote the maximum number of function evaluations and maximum runtime, respectively. If maxRuntime < inf, the metaheuristic is terminated after maxRuntime seconds; otherwise, the metaheuristic is terminated after calculating maxFE solutions' objective values. Usually, the number of iterations (i.e., generations) of metaheuristics equals to maxFE/N. For example,

```
platemo('Problem',@DTLZ1,'maxFE',20000);
```

sets the maximum number of function evaluations to 20000, and

```
platemo('Problem',@MaF1,'maxRuntime',10);
```

sets the maximum runtime to 10 s.

- 'save' denotes the number of saved solution sets, where the solution sets are saved to a file if save is positive, and the last solution set is displayed in a figure if save is negative. Details of the saved files are introduced in Section 5.2. For example,

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

displays the last solution set since the default value of save is −10, and

```
for i = 1 : 10
  platemo('algorithm',@GA,'save',5);
end
```

executes the genetic algorithm for ten runs and saves five solution sets obtained in each run to a unique file.

- 'outputFcn' denotes the function called before each iteration of the metaheuristic. An outputFcn should have two inputs (denoting the metaheuristic and the problem) and no output. For example, the default outputFcn displays or saves the optimization result according to the value of save.
- The remaining variables listed in Table 1 are used to define a new problem. The way of defining problems is introduced in Section 3.2.

PlatEMO defines each metaheuristic and problem as a class, which contains several properties denoting its parameters and methods denoting its functions (e.g., objective calculation and constraint calculation). In the next section, we elaborate on the way of defining new problems, while the way of defining new metaheuristics is not involved in this paper.

3. Defining an optimization problem in PlatEMO

3.1. Definition of optimization problems

An optimization problem can be mathematically defined as

$$\begin{aligned} \text{Min} \quad & f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x}) \\ \text{where} \quad & \mathbf{x} = (x_1, x_2, \ldots, x_D) \in \Omega, \\ \text{s.t.} \quad & g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots \leq 0 \end{aligned} \tag{1}$$

where $\mathbf{x}$ is a solution (or decision vector) consists of D decision variables, $f_1, f_2, \ldots$ are M objective functions, $g_1, g_2, \ldots$ are constraint functions, and Ω is the decision space (or search space) determined by the encoding scheme, lower bounds, and upper bounds. It should be noted that the above problem definition cannot exactly describe a few types of problems that can be solved by PlatEMO, such as dynamic optimization problems, multitasking optimization problems, and bilevel optimization problems, where the detailed definitions of these types of problems are referred to [23–25].

The goal of solving an optimization problem is to find a set of feasible and optimal solutions, where a feasible solution means that all the constraints are satisfied, and an optimal solution means that all the objectives are minimized and all the constraints are satisfied. Some important notes are presented as follows.

- The real optimal solutions cannot be found, since optimization problems are generally NP-hard. Thus, it cannot judge how good a solution is, but can only judge whether a solution is better than another one.
- All the objectives are to be minimized, that is, the maximization of an objective should be converted into the minimization of its negative.
- Equality constraints cannot be satisfied, since the probability of obtaining an exact decision variable value is zero in a stochastic optimization process. The way of relaxing equality constraints is introduced in Section 3.2.
- A metaheuristic generally outputs a set of N solutions, but it does not guarantee the feasibility and optimality of each solution.
- For single-objective optimization (i.e., $M = 1$), the solution satisfying all the constraints and having the minimum value of f_1 in the found solution set is the best one.
- For multi-objective optimization (i.e., $M \geqslant 2$), the feasible and non-dominated solutions in the found solution set are the best ones. A non-dominated solution means that it is not Pareto dominated by any others in the solution set, where solution $\mathbf{x}$ Pareto dominates solution $\mathbf{y}$ if the following conditions satisfy:

$$\begin{cases} f_i(\mathbf{x}) \leqslant f_i(\mathbf{y}), & \forall i = 1, \ldots, M \\ f_j(\mathbf{x}) < f_j(\mathbf{y}), & \exists j = 1, \ldots, M \end{cases}. \tag{2}$$

All the non-dominated solutions are with the same quality, and any one of them can be selected.

Fig. 5 illustrates the definition of a 2-objective 10-variable optimization problem, which includes the components that should be clarified and defined in PlatEMO. Moreover, four images of the problem are also plotted, including the landscape, the Pareto set[1], the Pareto front[2], and the feasible region. Note that the landscape is in the function space (i.e., the X-axis denotes x_2 and the Y-axis denotes f_1), the Pareto set is in the decision space and plotted by parallel coordinates [26] (i.e., the X-axis denotes each dimension

[1] The set of real optimal solutions for multi-objective optimization, where a real optimal solution is not Pareto dominated by any others in the whole decision space.
[2] The objective values of real optimal solutions.

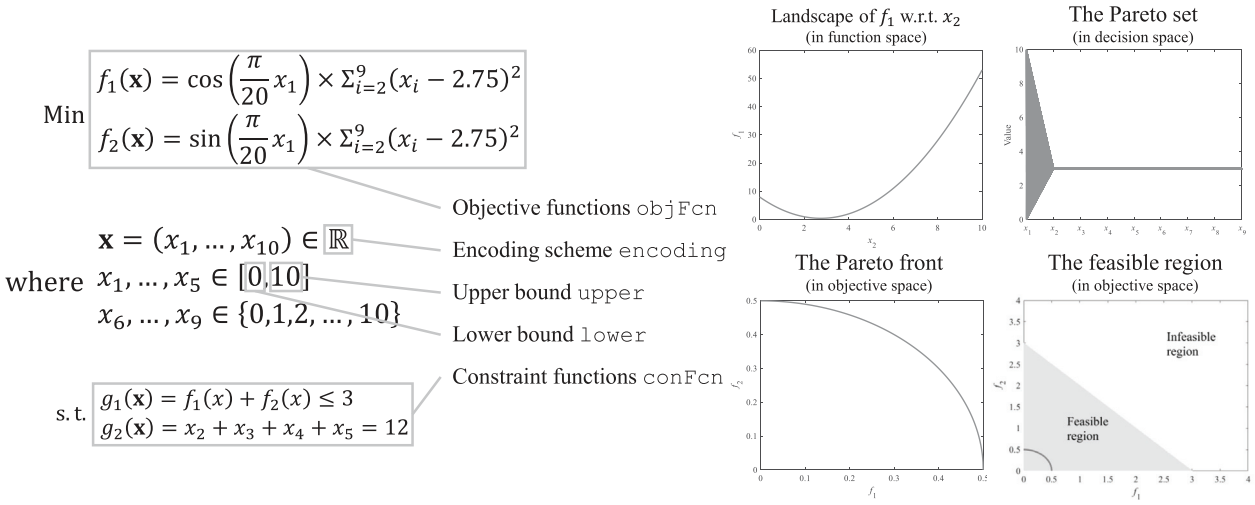

Fig. 5. An illustrative example of an optimization problem and its images.

of a decision vector and the Y-axis denotes the value on this dimension), and the Pareto front and the feasible region are in the objective space (i.e., the X-axis denotes f_1 and the Y-axis denotes f_2).

3.2. Defining problems in command mode and application module

When using the command mode of PlatEMO, users can define a problem by specifying the variables listed in the bottom half of Table 1, where the details of these variables are described as follows.

- '`objFcn`' denotes the objective functions, whose value should be a function handle or a cell. Note that an objective indicates a MATLAB function rather than a mathematical function, where any MATLAB function with one input and one output is suitable, and the input should be a decision vector and the output should be the objective value. All the objectives are to be minimized. Taking the problem illustrated in Fig. 5 as an example,

```
h=@(x)sum((x(2:end)-2.75).^2);
f1=@(x)cos(pi/20*x(1))*h(x);
platemo('objFcn',f1);
```

minimizes one objective $f_1(\mathbf{x})$, and

```
h=@(x)sum((x(2:end)-2.75).^2);
f1=@(x)cos(pi/20*x(1))*h(x);
f2=@(x)sin(pi/20*x(1))*h(x);
platemo('objFcn',{f1,f2});
```

minimizes two objectives $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$. Here `f1` and `f2` are set to anonymous functions, and they can also be set to the names of existing `.m` files. The number of objectives `M` is automatically determined according to `objFcn`.

- '`encoding`' denotes the encoding scheme of solutions, which should be a row vector of 1 (real variable), 2 (integer variable), 3 (label variable), 4 (binary variable), or 5 (permutation variable). Hybrid encoding scheme can be achieved by setting different values in the vector. According to Fig. 5,

```
h=@(x)sum((x(2:end)-2.75).^2);
f1=@(x)cos(pi/20*x(1))*h(x);
f2=@(x)sin(pi/20*x(1))*h(x);
platemo('objFcn',{f1,f2},...
'encoding',[1,1,1,1,1,2,2,2,2]);
```

specifies five real variables $x_1, \ldots, x_5$ and four integer variables $x_6, \ldots, x_9$. The number of decision variables `D` is automatically determined according to `encoding`.

- '`lower`' and '`upper`' denote the lower and upper bounds of solutions, respectively, which are meaningful for real and integer variables. '`lower`' and '`upper`' should be row vectors with the same length as `encoding`. According to Fig. 5,

```
h=@(x)sum((x(2:end)-2.75).^2);
f1=@(x)cos(pi/20*x(1))*h(x);
f2=@(x)sin(pi/20*x(1))*h(x);
platemo('objFcn',{f1,f2},...
'encoding',[1,1,1,1,1,2,2,2,2],...
'lower',zeros(1,9),'upper',10*ones(1,9));
```

sets the decision space to $[0, 10]^9$.

- '`conFcn`' denotes the constraint functions, which has the same form as `objFcn`. A constraint is satisfied only when the function value is not positive. According to Fig. 5,

```
h=@(x)sum((x(2:end)-2.75).^2);
f1=@(x)cos(pi/20*x(1))*h(x);
f2=@(x)sin(pi/20*x(1))*h(x);
g1=@(x)f1(x)+f2(x)-3;
platemo('objFcn',{f1,f2},'conFcn',g1,...
'encoding',[1,1,1,1,1,2,2,2,2],...
'lower',zeros(1,9),'upper',10*ones(1,9));
```

adds one constraint $g_1(\mathbf{x})$. For equality constraints like $x_2 + x_3 + x_4 + x_5 = 12$, it can be eliminated by optimizing only

x_2, x_3, x_4 and letting $x_5 = 12 - x_2 - x_3 - x_4$. Besides, an inequality constraint should be added to make $x_5 \geqslant 0$, that is,

```
h=@(x)sum((x(2:end)-2.75).^2)+...
(12-x(2)-x(3)-x(4)-2.75).^2;
f1=@(x)cos(pi/20*x(1))*h(x);
f2=@(x)sin(pi/20*x(1))*h(x);
g1=@(x)f1(x)+f2(x)-3;
g2=@(x)x(2)+x(3)+x(4)-12
platemo('objFcn',{f1,f2},'conFcn',{g1,g2},...
'encoding',[1,1,1,1,2,2,2,2],...
'lower',zeros(1,8),'upper',10*ones(1,8));
```

converts the equality constraint into an inequality constraint $g_2(\mathbf{x})$. For general equality constraints $g(\mathbf{x}) = 0$, it should be relaxed to an inequality constraint $|g(\mathbf{x})| \leqslant \epsilon$, where $\epsilon > 0$ is a tiny tolerance value. For example,

```
h=@(x)sum((x(2:end)-2.75).^2);
f1=@(x)cos(pi/20*x(1))*h(x);
f2=@(x)sin(pi/20*x(1))*h(x);
g1=@(x)f1(x)+f2(x)-3;
g3=@(x)abs(x(2)+x(3)+x(4)+x(5)-12)-0.1
platemo('objFcn',{f1,f2},'conFcn',{g1,g3},...
'encoding',[1,1,1,1,1,2,2,2,2],...
'lower',zeros(1,9),'upper',10*ones(1,9));
```

relaxes the equality constraint into an inequality constraint $g_3(\mathbf{x})$ with $\epsilon = 0.1$.

- `'decFcn'` denotes the function for repairing an invalid solution[3], which is called before objective and constraint calculation. A repair function should have one input and one output, where the input is a decision vector and the output is the repaired decision vector. The default repair function repairs solutions according to its encoding scheme (e.g., making the real decision variables always in between `lower` and `upper`), and a new repair function can be defined for constructing a special decision space. For example,

```
h=@(x)sum((x(2:end)-2.75).^2);
f1=@(x)cos(pi/20*x(1))*h(x);
f2=@(x)sin(pi/20*x(1))*h(x);
g1=@(x)f1(x)+f2(x)-3;
p=@(x)min(max([x(1),x(2:5)./sum(x(2:...
5))*12,round(x(6:end))],0),10);
platemo('objFcn',{f1,f2},'conFcn',g1,...
'encoding',[1,1,1,1,1,2,2,2,2],'decFcn',p,...
'lower',zeros(1,9),'upper',10*ones(1,9));
```

defines a repair function $p(\mathbf{x})$ making $x_2 + x_3 + x_4 + x_5 = 12$ always holds, where no constraint needs to be added.

- `'evalFcn'` denotes the function for evaluating a solution, where `'objFcn'`, `'conFcn'`, and `'decFcn'` are called by it. An evaluation function should have one input and three outputs, where the input is a decision vector and the outputs include the repaired decision vector, the objective values, and the constraint values. It can be found from the above codes that, the function $h(\mathbf{x})$ of each solution is calculated four times within $f_1(\mathbf{x}), f_2(\mathbf{x})$, and $g_1(\mathbf{x})$, which is highly redundant. To improve the efficiency, users can define `'evalFcn'` instead of `'objFcn'`, `'conFcn'`, and `'decFcn'` like

```
function [x,f,g] = Eval(x)
  x(2:5)=x(2:5)./sum(x(2:5))*12;
  x(6:end)=round(x(6:end));
  x=min(max(x,0),10);
  h=sum((x(2:end)-2.75).^2);
  f(1)=cos(pi/20*x(1))*h;
  f(2)=sin(pi/20*x(1))*h;
  g=f(1)+f(2)-3;
end
```

and then call

```
platemo('evalFcn',@Eval,'M',2,...
'encoding',[1,1,1,1,1,2,2,2,2],...
'lower',zeros(1,9),'upper',10*ones(1,9));
```

Note that the number of objectives `M` should be specified when specifying `evalFcn` instead of `objFcn`.

- `'initFcn'` denotes the function for initializing a solution set, which is usually called by metaheuristic at the beginning. An initialization function should have one input and one output, where the input is the number of solutions and the output is a matrix with each row being a decision vector. The default initialization function uses different strategies to randomly initialize solutions for different encoding schemes, and a new initialization function can be defined to heuristically initialize solutions. For example,

```
q=@(N)[rand(N,1)*10,rand(N,8)*5];
platemo('evalFcn',@Eval,'M',2,...
'encoding',[1,1,1,1,1,2,2,2,2],'initFcn',q,...
'lower',zeros(1,9),'upper',10*ones(1,9));
```

defines an initialization function $q(N)$ initializing solutions within $[0,10] \times [0,5]^8$ rather than the original decision space $[0,10]^9$, which can accelerate the convergence.

- `'objGradFcn'` denotes the functions for calculating the gradients of objectives, which is called by gradient-assisted optimizers. A gradient function should have one input and one output, where the input is a decision vector and the output is the gradient. The default gradient functions estimate the gradients by using forward finite difference. For example,

[3] An *invalid* solution indicates that its objectives and constraints cannot be calculated, while an *infeasible* solution indicates that at least one of its constraints is not satisfied.

```
h=@(x)sum((x(2:end)-2.75).^2);
fg1=@(x)[-pi/20*sin(pi/20*x(1))*h(x),...
cos(pi/20*x(1))*2*(x(2:end)-2.75)];
fg2=@(x)[pi/20*cos(pi/20*x(1))*h(x),...
sin(pi/20*x(1))*2*(x(2:end)-2.75)];
platemo('evalFcn',@Eval,'M',2,...
'encoding',[1,1,1,1,1,2,2,2,2],...
'lower',zeros(1,9),'upper',10*ones(1,9),...
'objGradFcn',{fg1,fg2});
```

defines the gradient functions $fg_1(\mathbf{x})$ and $fg_2(\mathbf{x})$ of the two objectives.

- '`conGradFcn`' denotes the functions for calculating the gradients of constraints, which has the same form as `objGradFcn`. For example,

```
h=@(x)sum((x(2:end)-2.75).^2);
fg1=@(x)[-pi/20*sin(pi/20*x(1))*h(x),...
cos(pi/20*x(1))*2*(x(2:end)-2.75)];
fg2=@(x)[pi/20*cos(pi/20*x(1))*h(x),...
sin(pi/20*x(1))*2*(x(2:end)-2.75)];
gg1=@(x)fg1(x)+fg2(x);
platemo('evalFcn',@Eval,'M',2,...
'encoding',[1,1,1,1,1,2,2,2,2],...
'lower',zeros(1,9),'upper',10*ones(1,9),...
'conGradFcn',gg1);
```

defines the gradient function $gg_1(\mathbf{x})$ of the constraint.

- '`data`' denotes a data that is involved in all the defined functions, which can be any type of variable. If `data` is specified, all the defined functions should have two inputs, where the first input is the original one and the second input is `data`. For example,

```
q=@(N,d)d(1:N,:);
rs=RandStream('mlfg6331_64','Seed',0);
platemo('evalFcn',@(x,d)Eval(x),'M',2,...
'encoding',[1,1,1,1,1,2,2,2,2],...
'lower',zeros(1,9),'upper',10*ones(1,9)),...
'data',[rand(rs,100,1)*10,rand(rs,100,8)*5];
```

fixes the initialized solutions by using an additional random number stream `rs` with a fixed seed.

When using the application module of the GUI, users can also define a problem by specifying the same variables introduced above. As shown in the left panel of Fig. 3, users should define these variables in the textboxes rather than writing codes. The application module provides many templates and guidance to facilitate the defining of problems, and users can additionally save, load, and validate the defined problem.

3.3. Defining problems in test module and experiment module

When using the test module or experiment module of the GUI, users should write the problem as a class (a `.m` file), which is the same as all the existing problems provided by PlatEMO. A problem class should be derived from the class `PROBLEM`, which contains all the variables introduced in Section 3.2 as well as more delicate methods. As listed in Table 2, a problem class can override several methods of `PROBLEM` like.

```
classdef myProblem < PROBLEM
  methods
    function Setting(obj)
      %...  ...
    end
    function P = Initialization(obj,N)
      %...  ...
    end
    function P = Evaluation(obj,Dec)
      %...  ...
    end
    function Dec = CalDec(obj,Dec)
      %...  ...
    end
    function Obj = CalObj(obj,Dec)
      %...  ...
    end
    function Con = CalCon(obj,Dec)
      %...  ...
    end
    function J = CalObjGrad(obj,x)
      %...  ...
    end
    function J = CalConGrad(obj,x)
      %...  ...
    end
    function R = GetOptimum(obj,N)
      %...  ...
    end
    function R = GetPF(obj)
      %...  ...
    end
    function score=CalMetric(obj,met,P)
      %...  ...
    end
    function DrawDec(obj,P)
      %...  ...
    end
    function DrawObj(obj,P)
      %...  ...
    end
  end
end
```

where the details are described as follows.

- `Setting()` sets the properties of the problem, including the solution set size `N`, the number of objectives `M`, the number of decision variables `D`, the maximum number of function evaluations `maxFE`, the maximum runtime `maxRuntime`, the encoding scheme `encoding`, the lower bounds `lower`, and the upper bounds `upper`. This method does not have input or output. This method is called before optimization, and the settings in this method override those specified by users. For the problem illustrated in Fig. 5,

Table 2
Methods that can be overridden in a problem class.

Method	Input	Output	Description
`Setting()`	None	None	Setting the properties of the problem before optimization
`Initialization()`	Number of initialized solutions `N`	Initial solution set `P`	Initializing a solution set
`Evaluation()`	A variable matrix `Dec` with each row being a decision vector	A solution set `P`	Evaluating solutions
`CalDec()`	A variable matrix `Dec` with each row being a decision vector	Repaired variable matrix `Dec`	Repairing invalid solutions
`CalObj()`	A variable matrix `Dec` with each row being a decision vector	An objective matrix `Obj` with each row being the objective values of a solution	The objective functions
`CalCon()`	A variable matrix `Dec` with each row being a decision vector	A constraint matrix `Con` with each row being the constraint values of a solution	The constraint functions
`CalObjGrad()`	A decision vector `x`	A Jacobian matrix `J`	Calculating the gradients of objectives
`CalConGrad()`	A decision vector `x`	A Jacobian matrix `J`	Calculating the gradients of constraints
`GetOptimum()`	Number of optimal points `N`	Optimal point set `R`	Generating optimal objective values of the problem
`GetPF()`	None	Data `R` for plotting the Pareto front	Generating the Pareto front of the problem
`CalMetric()`	Metric name `met` and a solution set `P`	Metric value `score`	Calculate a metric value
`DrawDec()`	A solution set `P`	None	Plotting a solution set in the decision space
`DrawObj()`	A solution set `P`	None	Plotting a solution set in the objective space

```
function Setting(obj)
  obj.M=2;
  obj.D=9;
  obj.encoding=[1,1,1,1,1,2,2,2,2];
  obj.lower=zeros(1,obj.D);
  obj.upper=10*ones(1,obj.D);
end
```

sets the number of objectives, the number of decision variables, the encoding scheme, the lower bounds, and the upper bounds.

- `Initialization()` is similar to `initFcn` in Table 1 for initializing a solution set, where the main difference is that the output of `Initialization()` is a set of solution objects `P`. A solution object stores the decision vector, objective values, and constraint values of a solution, which is obtained by calling the method `Evaluation()`. For example,

```
function P = Initialization(obj,N)
  if nargin<2
    N=obj.N;
  end
  Dec=[rand(N,1)*10,rand(N,8)*5];
  P=obj.Evaluation(Dec);
end
```

Note that a metaheuristic may call `obj.Initialization()` rather than `obj.Initialization(N)`, hence `N` should be set to the solution set size `obj.N` when it is not specified.

- `Evaluation()` is similar to `evalFcn` in Table 1 for evaluating solutions, where the main difference is that the input of `Evaluation()` is the decision vectors of multiple solutions, and the output of `Evaluation()` is a set of solution objects `P`. For example,

```
function P = Evaluation(obj,varargin)
  Dec=varargin{1};
  Dec(:,2:5)=Dec(:,2:5)./sum(Dec(:,2:5),2)*12;
  Dec(:,6:end)=round(Dec(:,6:end));
  Dec=min(max(Dec,0),10);
  h=sum((Dec(:,2:end)-2.75).^2,2);
  Obj(:,1)=cos(pi/20*Dec(:,1)).*h;
  Obj(:,2)=sin(pi/20*Dec(:,1)).*h;
  Con=Obj(:,1)+Obj(:,2)-3;
  P=SOLUTION(Dec,Obj,Con,varargin{2:end});
  obj.FE=obj.FE+length(P);
end
```

instantiates solutions via the class `SOLUTION`, and increases the number of consumed function evaluations `obj.FE` accordingly. Note that this method receives multiple inputs via `varargin`, where the decision vectors are stored in the first input `varargin{1}` and the additional vectors are stored in the second input `varargin{2}`. If this method receives only the decision vectors, some metaheuristics using additional vectors (e.g., the particle swarm optimization algorithm [27] using velocity) cannot be used.

- `CalDec()` is similar to `decFcn` in Table 1 for repairing invalid solutions, where the main difference is that the input of `CalDec()` is the decision vectors of multiple solutions, and the output of `CalDec()` is the repaired decision vectors of multiple solutions. If users do not define `Evaluation()`, they can define

```
function Dec = CalDec(obj,Dec)
  Dec(:,2:5)=Dec(:,2:5)./sum(Dec(:,2:5),2)*12;
  Dec(:,6:end)=round(Dec(:,6:end));
  Dec=min(max(Dec,0),10);
end
```

to repair multiple solutions simultaneously.

- `CalObj()` is similar to `objFcn` in Table 1 for calculating the objective values of solutions, where the main difference is that the input of `CalObj()` is the decision vectors of multiple solutions, and the output of `CalObj()` is the objective values of multiple solutions. If users do not define `Evaluation()`, they can define

```
function Obj = CalObj(obj,Dec)
  h=sum((Dec(:,2:end)-2.75).^2,2);
  Obj(:,1)=cos(pi/20*Dec(:,1)).*h;
  Obj(:,2)=sin(pi/20*Dec(:,1)).*h;
end
```

to calculate the objective values of multiple solutions simultaneously.

- `CalCon()` is similar to `conFcn` in Table 1 for calculating the constraint values of solutions, where the main difference is that the input of `CalCon()` is the decision vectors of multiple solutions, and the output of `CalCon()` is the constraint values of multiple solutions. If users do not define `Evaluation()`, they can define

```
function Con = CalCon(obj,Dec)
  Obj=obj.CalObj(Dec);
  Con=Obj(:,1)+Obj(:,2)-3;
end
```

to calculate the constraint values of multiple solutions simultaneously.

- `CalObjGrad()` is similar to `objGradFcn` in Table 1 for calculating the gradients of objectives, where the main difference is that the output of `CalObjGrad()` is a Jacobian matrix with each row being the gradient on an objective. For example,

```
function J = CalObjGrad(obj,x)
  h=sum((Dec(:,2:end)-2.75).^2,2);
  J=[-pi/20*sin(pi/20*x(1))*h,...
  cos(pi/20*x(1))*2*(x(2:end)-2.75);...
  pi/20*cos(pi/20*x(1))*h,...
  sin(pi/20*x(1))*2*(x(2:end)-2.75)];
end
```

- `CalConGrad()` is similar to `conGradFcn` in Table 1 for calculating the gradients of constraints, where the main difference is that the output of `CalConGrad()` is a Jacobian matrix with each row being the gradient on a constraint. For example,

```
function J = CalConGrad(obj,x)
  h=sum((Dec(:,2:end)-2.75).^2,2);
  J=[-pi/20*sin(pi/20*x(1))*h+...
  pi/20*cos(pi/20*x(1))*h,...
  cos(pi/20*x(1))*2*(x(2:end)-2.75)+...
  sin(pi/20*x(1))*2*(x(2:end)-2.75)];
end
```

- `GetOptimum()` generates the real optimal objective values of the problem, which is meaningful for only multi-objective optimization. This method should have one input and one output, where the input is the number of optimal points `N` and the output is a set of uniformly distributed optimal points `R` on the Pareto front. This method is called before optimization, and the generated optimal points will be stored in the property `obj.optimum` for performance metric calculation. For the problem illustrated in Fig. 5,

```
function R = GetOptimum(obj,N)
  R=UniformPoint(N,obj.M);
  R=0.5*R./sqrt(sum(R.^2),2);
end
```

generates approximately `N` optimal points on the Pareto front $f_1^2+f_2^2=0.5$ via the strategies introduced in [28]. Besides, for multi-objective optimization problems whose Pareto fronts are unknown, a single reference point can be generated for HV calculation, where the details are introduced in Section 5.3.

- `GetPF()` generates the Pareto front of the problem, which is meaningful for only two- and three-objective optimization. This method should have no input and one output, where the output is the data `R` for plotting the Pareto front. This method is called before optimization, and the generated data will be stored in the property `obj.PF` for visualization. For the problem illustrated in Fig. 5,

```
function R = GetPF(obj)
  R=obj.GetOptimum(100);
end
```

generates the curve of the Pareto front, which can be used in the method `DrawObj()` to plot the Pareto front shown in Fig. 5. The difference between `obj.optimum` and `obj.PF` is illustrated in Fig. 6, where the `obj.optimum` of 3-objective DTLZ2 [19] contains a set of uniformly distributed optimal points and the `obj.PF` is the whole surface represented by many curves. Moreover, this method can also generate the feasible region instead of the Pareto front for constrained two-objective optimization, for example,

```
function R = GetPF(obj)
  [x,y]=meshgrid(0:0.01:4);
  z=nan(size(x));
  fes=x+y<=3;
  z(fes)=0;
  R={x,y,z};
end
```

generates the data of the feasible region, which can be used in the method `DrawObj()` to plot the feasible region shown in Fig. 5.

- `CalMetric()` calculate a metric value of a solution set. This method should have two inputs and one output, where the first input is a metric name `met`, the second input is a set of solution objects `P`, and the output is the metric value. This method is called in the GUI, and can be overridden for feeding different variables to specific metric functions, for example,

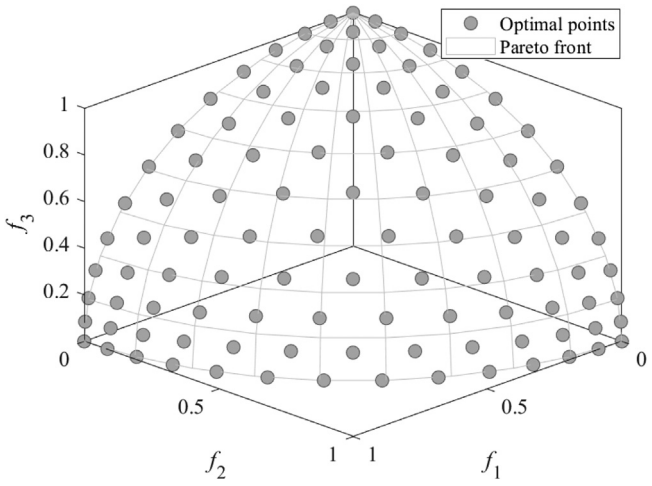

Fig. 6. The optimal points `optimum` and Pareto front `PF` of 3-objective DTLZ2.

```
function score = CalMetric(obj,met,P)
  switch met
    case 'HV'
      score=feval(met,P,[2 2]);
    otherwise
      score=feval(met,P,obj.optimum);
  end
end
```

set the reference point for HV calculation to `[2,2]` rather than the real optimal objective values `optimum`.

- `DrawDec()` plots a solution set in the decision space. This method should have one input and no output, where the input is a set of solution objects `P`. This method is called in the GUI, and can be overridden for personalized visualization, for example,

```
function DrawDec(obj,P)
  Draw(P.decs);
  Draw(P.best.decs,'-k','LineWidth',2);
end
```

plots a solution set in the decision space and highlights the feasible and non-dominated solutions.

- `DrawObj()` plots a solution set in the objective space, which is meaningful for only multi-objective optimization. This method should have one input and no output, where the input is a set of solution objects `P`. This method is called in the GUI, and can be overridden for personalized visualization, for example,

```
function DrawObj(obj,P)
  Draw(P.objs);
  Draw(P.best.objs,'*k');
end
```

plots a solution set in the objective space and highlights the feasible and non-dominated solutions.

After constructing the class and defining the above methods, the problem can be selected and solved in the test module and experiment module. Besides, the problem can also be solved in the command mode like

```
platemo('problem',@myProblem);
```

To better illustrate the functions called in the execution of the above command, Fig. 7 plots a sequence diagram of solving `@myProblem` via the default metaheuristic.

4. Selecting metaheuristics in PlatEMO

4.1. Labels for characterizing problems

A metaheuristic can be selected by mouse-clicks when using the GUI and by `platemo('algorithm',@GA)` when using the command mode. While PlatEMO currently provides more than 200 metaheuristics, it is not straightforward to select a suitable one for a given problem [29]. To address this issue, PlatEMO tags each metaheuristic with multiple labels, where each label indicates a type of optimization problem that can be handled by the metaheuristic [30]. After defining a problem, users should determine the labels of the problem, then select a metaheuristic having the same labels to solve the problem. As illustrated in Table 3, these labels cover various types of optimization problems, where each one is in fact a research area in the computational intelligence [4]. The details of these labels are as follows.

- `<single>` denotes single-objective optimization [7], which indicates that the problem has a single objective to be minimized.
- `<multi>` denotes multi-objective optimization [31], which indicates that the problem has multiple objectives to be optimized simultaneously. In particular, it indicates that the problem has two or three objectives in PlatEMO.
- `<many>` denotes many-objective optimization [1], which indicates that the problem has four or more objectives to be optimized simultaneously. In general, the selection strategies in metaheuristics for multi-objective optimization and many-objective optimization are different.
- `<real>` denotes continuous optimization [32], where the decision variables are real numbers (e.g., the portfolio optimization problem [33]).
- `<integer>` denotes integer optimization [34], where the decision variables are integers (e.g., the aperture optimization problem [35]).
- `<label>` indicates that the decision variables are labels, such as the community detection problem [36]. Although labels are also represented by integers, they are not numerical and cannot be calculated.
- `<binary>` denotes binary optimization [37], which indicates that the decision variables are binary numbers (e.g., the feature selection problem [38]).
- `<permutation>` denotes routing optimization [39], which indicates that each decision vector is a permutation (e.g., the vehicle routing problem [40]).
- `<large>` denotes large-scale optimization [3], which indicates that the problem has more than 100 decision variables. To accelerate the convergence, special variation operators are designed to tackle the curse of dimensionality [5].
- `<constrained>` denotes constrained optimization [41], which indicates that the problem has at least one constraint. To consider constraints in the optimization process, special

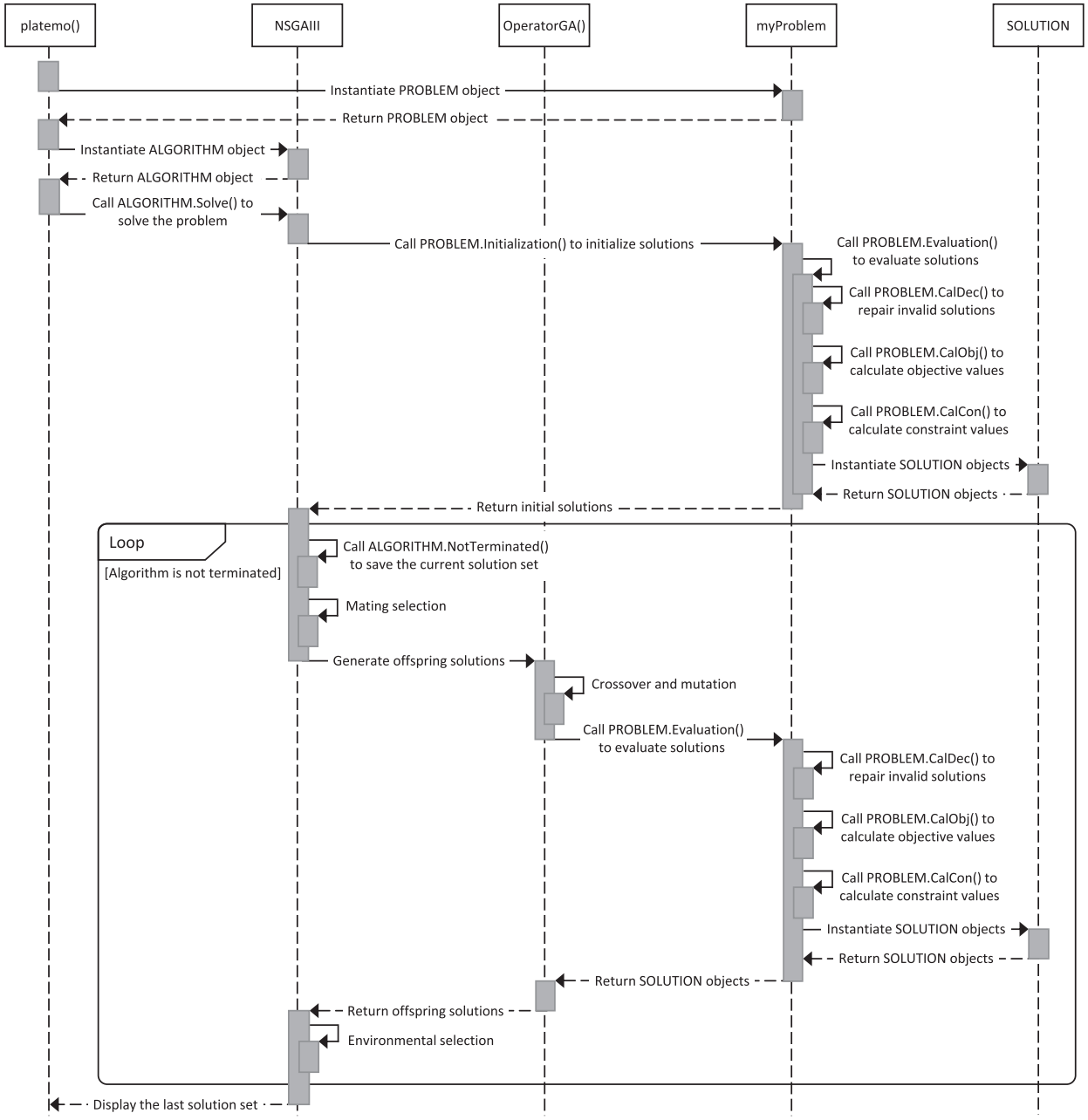

Fig. 7. Sequence diagram of solving a problem defined by a class via PlatEMO.

selection strategies are designed to balance between objective optimization and constraint satisfaction [42].

- <expensive> denotes expensive optimization [2], which indicates that the objectives and constraints are computationally expensive. To save the expensive function evaluations, surrogate models are adopted to predict the qualities of candidate solutions [43].
- <multimodal> denotes multimodal optimization [44], which indicates that there exist multiple optimal solutions with similar objective values but totally different decision variables. To find all the equivalent optimal solutions, special selection strategies are designed to maintain the diversity of the solution set in the decision space [45].
- <sparse> denotes sparse optimization [46], which indicates that most decision variables of the optimal solutions are zero. To accelerate the convergence, special variation operators are designed to generate sparse solutions efficiently [47].
- <dynamic> denotes dynamic optimization [23], which indicates that the objectives and constraints vary periodically. To quickly react to changes, special initialization strategies are designed to accelerate the convergence in each time period [48].
- <multitask> denotes multitasking optimization [24], which indicates that there exist multiple tasks to be solved simultaneously. Each task is an independent optimization problem and all tasks share the same decision space and solution set. To solve multiple tasks in a single run, special search strategies are designed to transfer useful genetic material between tasks [49].
- <bilevel> denotes bilevel optimization [25], which indicates that there exist an upper-level problem and a lower-

Table 3
Labels for tagging each metaheuristic and problem in PlatEMO.

Number of objectives	
Label	Meaning
<single>	The problem has a single objective (i.e., $\mathtt{M}=1$)
<multi>	The problem has two or three objectives (i.e., $\mathtt{M}=2,3$)
<many>	The problem has four or more objectives (i.e., $\mathtt{M}\geqslant 4$)
Encoding scheme	
Label	Meaning
<real>	The decision variables are real numbers (e.g., $\mathbf{x}=(0.5,1.2,3.1,2.8,4.9)$)
<integer>	The decision variables are integers (e.g., $\mathbf{x}=(5,1,3,3,1)$)
<label>	The decision variables are labels (e.g., $\mathbf{x}=(1,1,1,2,2)$)
<binary>	The decision variables are binary numbers (e.g., $\mathbf{x}=(1,0,0,1,0)$)
<permutation>	The decision variables constitute a permutation (e.g., $\mathbf{x}=(5,1,2,3,4)$)
Special difficulty	
Label	Meaning
<large>	The problem has more than 100 decision variables
<constrained>	The problem has at least one constraint
<expensive>	The objectives and constraints are computationally expensive, i.e., only a few function evaluations are available
<multimodal>	There exist multiple optimal solutions with similar objective values but different decision variables, all of which are expected to be found
<sparse>	Most decision variables of the optimal solutions are zero
<dynamic>	The objectives and constraints vary periodically, the optimal solutions for each time period are expected to be found
<multitask>	The problem contains multiple tasks to be solved simultaneously
<bilevel>	The problem is a bilevel optimization problem
<robust>	The objectives and constraints are influenced by uncertain factors, robust solutions rather than optimal solutions are expected to be found
<none>	Empty label

level problem. The lower-level problem is nested within the upper-level problem, where a solution for the upper-level problem is feasible if and only if it is an optimal solution for the lower-level problem. To find the feasible and optimal solutions for the upper-level problem, special search frameworks are designed to optimize the two problems alternately [50].

- <robust> denotes robust optimization [51], which indicates that the objectives and constraints are influenced by uncertain factors. To find robust solutions rather than optimal solutions, special evaluation strategies are designed to balance between the quality and robustness of solutions [52].
- <none> denotes an empty label, the function of which is introduced in the next subsection.

4.2. Selecting metaheuristics with desired labels

PlatEMO tags each metaheuristic with multiple label sets to indicate its application scope, by means of the comment in the second line of the metaheuristic class. For example, if a metaheuristic is tagged with three label sets <single> <real> <constrained/none>, it means that the metaheuristic is suitable for solving single-objective continuous optimization problems with or without constraints. On the other hand, the label sets <single> <real> mean that the metaheuristic can only solve unconstrained optimization problems, the label sets <single> <real> <constrained> mean that the metaheuristic can only solve constrained optimization problems, and the label sets <single> <real/binary> mean that the metaheuristic can solve unconstrained optimization problems with either real variables or binary variables. As a consequence, the Cartesian product between all the label sets constitutes all the types of problems that can be solved by the metaheuristic. In particular, the label <none> is used together with other labels such as <large>, where <large> means that the metaheuristic can only solve large-scale optimization problems while <large/none> means that the metaheuristic can solve large- and small-scale optimization problems.

To identify the metaheuristics tagged with desired labels, users can select multiple labels in Step (1) of the test module, Step (3) of the application module, and Step (1) of the experiment module, then only the desired metaheuristics will be shown in the list to be selected. Besides, when using the command mode of PlatEMO, users should check the comment in the second line of each metaheuristic class by themselves.

5. Collecting optimization results from PlatEMO

5.1. Saving results in test module and application module

The output of a metaheuristic is generally a set of $\mathtt{N}$ solutions for the solved problem, where each solution consists of decision variables, objective values, and constraint values. After solving a problem using the test module or application module, the final solution set will be displayed on the axis in Figs. 2 and 3. In addition to saving the axis as an image file, the final solution set can be saved to a .txt, .dat, .csv, .mat, or .xlsx file, where each file stores a matrix with the following form:

$$\begin{bmatrix} x_{11} & \cdots & x_{1D} & f_{11} & \cdots & f_{1M} & g_{11} & \cdots \\ x_{21} & \cdots & x_{2D} & f_{21} & \cdots & f_{2M} & g_{21} & \cdots \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\ x_{N1} & \cdots & x_{ND} & f_{N1} & \cdots & f_{NM} & g_{N1} & \cdots \end{bmatrix}, \tag{3}$$

where x_{ij} denotes the j-th decision variable of the i-th solution, f_{ij} denotes the j-th objective value of the i-th solution, and g_{ij} denotes the j-th constraint value of the i-th solution.

To find the best decision vectors among the $\mathtt{N}$ solutions, users should select the feasible solution with the minimum objective value for single-objective optimization, and select the feasible and non-dominated solutions for multi-objective optimization.

For this aim, the method `best()` of the class `SOLUTION` can be called to obtain the best decision vectors, for example,

```
P=SOLUTION(Data(:,1:D),Data(:,D+1:D+M),...
 Data(:,D+M+1:end));
 P.best.decs
```

where `Data` is the matrix of a solution set with `D` decision variables and `M` objectives.

5.2. Saving results in command mode and experiment module

When solving a problem using the command mode of PlatEMO, the final solution set will be returned if the main function is called with outputs:

```
[Dec,Obj,Con]=platemo(...)
```

where `Dec` is a matrix consisting of the decision vectors, `Obj` is a matrix consisting of the objective values, and `Con` is a matrix consisting of the constraint values of the final solution set. If the main function is called without output, the obtained solution sets will be displayed in a figure if `save` is negative and saved to a `.mat` file if `save` is positive. The saved files are named as `'PlatEMO\Data\alg\alg_pro_M_D_run.mat'`, where `alg` is the name of the metaheuristic, `pro` is the name of the problem, `M` is the number of objectives, `D` is the number of decision variables, and `run` automatically increases from 1 until the file name does not exist. Each file saves a cell `result` and a struct `metric`, where `result` stores multiple solution sets and `metric` stores the performance metric values of the solution sets. More specifically, the whole optimization process is divided into `save` equal intervals, where the first column of `result` stores the number of consumed function evaluations at the last iteration of each interval, and the second column of `result` stores the solution set at the last iteration of each interval. Taking the file shown in Fig. 8 as an example, it is obtained by setting `save` to 5 and `maxFE` to 20000, hence five solution sets when the numbers of consumed function evaluations are 4000, 8000, 12000, 16000, and 20000 are stored in `result`, and the IGD metric values of the five solution sets are stored in `metric`.

Except for the total runtime `metric.runtime`, the metric values are not calculated or saved when using the command mode. By contrast, the metric values can be automatically calculated and saved in the experiment module, by means of selecting a performance metric in Step (7) of Fig. 4. Moreover, the number of saved results `save` and the path for saving the files can be set in Step (4) of Fig. 4.

5.3. Performance metrics

Performance metrics are used to quantitatively assess the quality of a solution set for a given problem, and are mainly tailored for multi-objective optimization due to the conflicting nature between

```
result =
  5×2 cell array                    {1×100 SOLUTION}
    {[ 4000]}    {1×100 SOLUTION}
    {[ 8000]}    {1×100 SOLUTION}
    {[12000]}    {1×100 SOLUTION}
    {[16000]}    {1×100 SOLUTION}
    {[20000]}    {1×100 SOLUTION}

metric =
  struct with fields:

    runtime: 1.1347
        IGD: [5×1 double]
```

Fig. 8. An illustrative example of two variables `result` and `metric` saved in a file.

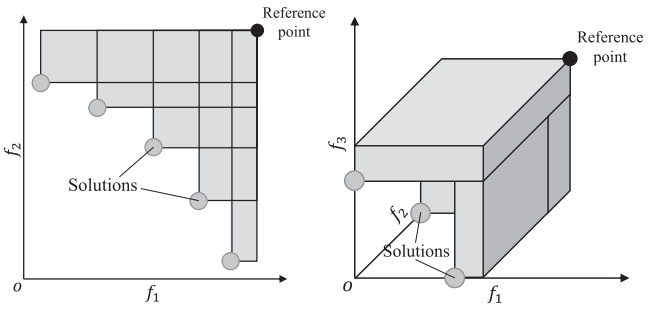

Fig. 9. Illustrative examples of HV calculation in two- and three-dimensional objective spaces, where the hypervolumes of the gray regions are the metric values.

objectives [53]. Each performance metric in PlatEMO is defined as a function (a `.m` file) and can be called like.

```
score=Pro.CalMetric(met,P);
```

where `Pro` is a problem class, `met` is a metric name, and `P` is a set of solution objects. Besides, the metric value `score` is a scalar, where the values of some metrics are the smaller the better and the values of the other metrics are the larger the better.

PlatEMO provides two performance metrics for single-objective optimization, including `Min_value` and `Feasible_rate`, where the former returns the minimum objective value among the feasible solutions in a solution set, and the latter returns the rate of feasible solutions in a solution set. On the other hand, PlatEMO provides various performance metrics to assess the convergence and/or diversity [54] of a solution set for multi-objective optimization, where `IGD` (i.e., inverted generational distance) [55] and `HV` (i.e., hypervolume) [56] are the most widely used ones.

`IGD` calculates the similarity between a solution set and a set of optimal points sampled on the Pareto front, where a smaller IGD value indicates a better convergence and diversity of the solution set. Note that IGD cannot be used when solving user-defined problems since the Pareto fronts as well as optimal points are unknown. By contrast, IGD can only be used when solving existing benchmark problems, whose Pareto fronts are known and the optimal solutions can be sampled by the strategies introduced in [28].

`HV` calculates the hypervolume of the region bounded by a solution set and a reference point, where a larger HV value indicates a better convergence and diversity of the solution set. As illustrated in Fig. 9, the calculation of HV is extremely complex since the bounded region is highly irregular, whose time complexity is exponentially related to the number of dimensions. Nevertheless, the calculation of HV requires only a reference point rather than a set of optimal solutions, where the reference point can be any point with slightly larger objective values than all the solutions. Therefore, HV is suggested to be used when solving user-defined multi-objective optimization problems. If the problem is defined in command mode or application module, the reference point is generated automatically. If the problem is defined in test module or experiment module, a proper reference point should be given as the output of the method `GetOptimum()`.

6. Conclusions

This paper presents a tutorial on solving optimization problems via PlatEMO, including the steps of defining optimization problems, selecting suitable metaheuristics, and collecting the optimization results. To define a problem, users should clarify its objective functions, constraint functions, encoding scheme, and

decision space, then specify the variables listed in Table 1 when using the command mode or application module, and specify the functions listed in Table 2 when using the test module or experiment module. To select a suitable metaheuristic, users should clarify the labels of the defined problem listed in Table 3, then select a metaheuristic tagged with the same labels. To collect the optimization results, users can save the final solution set as a matrix or image when using the test module or application module, save the final solution set as a set of `SOLUTION` objects when using the command mode or experiment module, and save the statistical results as a table when using the experiment module.

It is worth noting that the other functions (e.g., defining a new metaheuristic or performance metric) as well as the basic architecture of PlatEMO are not introduced in this paper, where the details can be found in the user manual of PlatEMO. To further enlarge the application scope of PlatEMO, more metaheuristics will be added in the future, such as noisy optimization [57], multi-fidelity optimization [58], and interval optimization [59].

CRediT authorship contribution statement

Ye Tian: Writing – original draft, Writing – review & editing. **Weijian Zhu:** Software, Validation. **Xingyi Zhang:** Investigation, Resources. **Yaochu Jin:** Supervision, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by the National Key R&D Program of China (No. 2018AAA0100100), and in part by the National Natural Science Foundation of China (No. 61876162, No. 61906001, No. 62136008, No. U20A20306, No. U21A20512).

References

[1] K. Li, R. Wang, T. Zhang, H. Ishibuchi, Evolutionary many-objective optimization: A comparative study of the state-of-the-art, IEEE Access 6 (2018) 26194–26214.
[2] Y. Jin, H. Wang, T. Chugh, D. Guo, K. Miettinen, Data-driven evolutionary optimization: An overview and case studies, IEEE Trans. Evol. Comput. 22 (3) (2019) 442–458.
[3] J. Jian, Z. Zhan, J. Zhang, Large-scale evolutionary optimization: a survey and experimental comparative study, Int. J. Mach. Learn. Cybern. 11 (2020) 729–745.
[4] J.D. Ser, E. Osaba, D. Molina, X. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P.N. Suganthan, C.A.C. Coello, F. Herrera, Bio-inspired computation: Where we stand and what's next, Swarm Evol. Comput. 48 (2019) 220–250.
[5] Y. Tian, L. Si, X. Zhang, R. Cheng, C. He, K.C. Tan, Y. Jin, Evolutionary large-scale multi-objective optimization: A survey, ACM Comput. Surv. 54 (8) (2022) 1–34.
[6] Y. Tian, H. Chen, X. Xiang, H. Jiang, X. Zhang, A comparative study on evolutionary algorithms and mathematical programming methods for continuous optimization, in: Proceedings of the 2022 IEEE Congress on Evolutionary Computation, 2022.
[7] S. Li, Y. Li, Y. Lin, Intelligent Optimization Algorithms and Emergent Computation, Tsinghua University Press, Beijing, 2019.
[8] S. Kenneth, Metaheuristics–the metaphor exposed, Int. Trans. Oper. Res. 22 (2015) 3–18.
[9] P. Kerschke, H.H. Hoos, F. Neumann, H. Trautmann, Automated algorithm selection: Survey and perspectives, Evol. Comput. 27 (1) (2019) 3–45.
[10] Q. Zhang, H. Li, MOEA/D: A multi-objective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731.
[11] Q. Zhang, A. Zhou, Y. Jin, RM-MEDA: A regularity model-based multiobjective estimation of distribution algorithm, IEEE Trans. Evol. Comput. 12 (1) (2008) 41–63.
[12] X. Zhang, Y. Tian, R. Cheng, Y. Jin, A decision variable clustering-based evolutionary algorithm for large-scale many-objective optimization, IEEE Trans. Evol. Comput. 22 (1) (2018) 97–112.
[13] Y. Tian, R. Cheng, X. Zhang, Y. Jin, PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum], IEEE Comput. Intell. Mag. 12 (4) (2017) 73–87.
[14] Y. Tian, R. Cheng, X. Zhang, Y. Jin, Techniques for accelerating multi-objective evolutionary algorithms in PlatEMO, in: Proceedings of the 2020 IEEE Congress on Evolutionary Computation, 2020.
[15] C.A.C. Coello, G.B. Lamont, D.A.V. Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, Springer, New York, 2007.
[16] C.A.C. Coello, S.G. Brambila, J.F. Gamboa, M.G.C. Tapia, R.H. Gómez, Evolutionary multiobjective optimization: open research areas and some challenges lying ahead, Complex Intell. Syst. 6 (2020) 221–236.
[17] Y. Tian, S. Yang, X. Zhang, Y. Jin, Using PlatEMO to solve multi-objective optimization problems in applications: A case study on feature selection, in: Proceedings of the 2019 IEEE Congress on Evolutionary Computation, 2019.
[18] J.H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, 1992.
[19] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable test problems for evolutionary multiobjective optimization, Evol. Multiobjective Optim. (2005) 105–145.
[20] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multi-objective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[21] S. Huband, P. Hingston, L. Barone, L. While, A review of multiobjective test problems and a scalable test problem toolkit, IEEE Trans. Evol. Comput. 10 (5) (2006) 477–506.
[22] R. Cheng, M. Li, Y. Tian, X. Zhang, S. Yang, Y. Jin, X. Yao, A benchmark test suite for evolutionary many-objective optimization, Complex Intell. Syst. 3 (1) (2017) 67–81.
[23] D. Yazdani, R. Cheng, D. Yazdani, J. Branke, Y. Jin, X. Yao, A survey of evolutionary continuous dynamic optimization over two decades-part A, IEEE Trans. Evol. Comput. 25 (4) (2021) 609–629.
[24] E. Osaba, J.D. Ser, A.D. Martinez, A. Hussain, Evolutionary multitask optimization: a methodological overview, challenges, and future research directions, Cogn. Comput. 14 (2022) 927–954.
[25] A. Sinha, P. Malo, K. Deb, Evolutionary bilevel optimization: An introduction and recent advances, Springer, Cham, 2017, Ch. Recent Advances in Evolutionary Multi-objective Optimization, pp. 71–103.
[26] M. Li, L. Zhen, X. Yao, How to read many-objective solution sets in parallel coordinates [educational forum], IEEE Comput. Intell. Mag. 12 (4) (2017) 88–100.
[27] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the 6th International Symposium on Micro Machine and Human Science, 1995, pp. 39–43.
[28] Y. Tian, X. Xiang, X. Zhang, R. Cheng, Y. Jin, Sampling reference points on the Pareto fronts of benchmark multi-objective optimization problems, in: Proceedings of the 2018 IEEE Congress on Evolutionary Computation, 2018.
[29] Y. Tian, S. Peng, X. Zhang, T. Rodemann, K.C. Tan, Y. Jin, A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks, IEEE Trans. Artif. Intell. 1 (1) (2020) 5–18.
[30] Y. Tian, M. Sun, S. Yang, X. Zhang, Tagging metaheuristics with problem-oriented labels for non-expert users, in: Proceedings of the 2022 IEEE Symposium Series on Computational Intelligence, 2022.
[31] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P.N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: A survey of the state of the art, Swarm Evol. Comput. 1 (1) (2011) 32–49.
[32] G.A. Trunfio, Big Data Optimization: Recent Developments and Challenges, Springer International Publishing Switzerland, 2016, Ch. Metaheuristics for continuous optimization of high-dimensional problems: State of the art and perspectives, pp. 437–460.
[33] A. Thakkar, K. Chaudhari, A comprehensive survey on portfolio optimization, stock price and trend prediction using particle swarm optimization, Arch. Comput. Methods Eng. 28 (2021) 2133–2164.
[34] B. Akay, D. Karaboga, B. Gorkemli, E. Kaya, A survey on the artificial bee colony algorithm variants for binary, integer and mixed integer programming problems, Appl. Soft Comput. 106 (2021) .
[35] Y. Tian, Y. Feng, C. Wang, R. Cao, X. Zhang, X. Pei, K.C. Tan, Y. Jin, A large-scale combinatorial many-objective evolutionary algorithm for intensity-modulated radiotherapy planning, IEEE Trans. Evol. Comput.
[36] Y. Tian, S. Yang, X. Zhang, An evolutionary multiobjective optimization based fuzzy method for overlapping community detection, IEEE Trans. Fuzzy Syst. 28 (11) (2020) 2841–2855.
[37] C. Qian, Y. Yu, Z.-H. Zhou, Subset selection by Pareto optimization, in: Advances in Neural Information Processing Systems, 2015, pp. 1774–1782.
[38] B.H. Nguyen, B. Xue, M. Zhang, A survey on swarm intelligence approaches to feature selection in data mining, Swarm Evol. Comput. 54 (2020) .
[39] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, A.A. Juan, Rich vehicle routing problem: Survey, ACM Comput. Surv. 47 (2) (2015) 1–28.
[40] Y. Tian, T. Zhang, J. Xiao, X. Zhang, Y. Jin, A coevolutionary framework for constrained multiobjective optimization problems, IEEE Trans. Evol. Comput. 25 (1) (2021) 102–116.
[41] J. Liang, X. Ban, K. Yu, B. Qu, K. Qiao, C. Yue, K. Chen, K.C. Tan, A survey on evolutionary constrained multi-objective optimization, IEEE Trans. Evol. Comput.
[42] Y. Wang, B. Wang, H. Li, G.G. Yen, Incorporating objective function information into the feasibility rule for constrained evolutionary optimization, IEEE Trans. Cybern. 46 (12) (2015) 1–15.
[43] R. Allmendinger, M.T. Emmerich, J. Hakanen, Y. Jin, E. Rigoni, Surrogate-assisted multicriteria optimization: Complexities, prospective solutions, and business case, J. Multi-Criteria Decis. Anal. 24 (1–2) (2017) 5–24.

[44] X. Li, M.G. Epitropakis, K. Deb, A. Engelbrecht, Seeking multiple solutions: An updated survey on niching methods and their applications, IEEE Trans. Evol. Comput. 21 (4) (2017) 518–538.
[45] Y. Tian, R. Liu, X. Zhang, H. Ma, K.C. Tan, Y. Jin, A multi-population evolutionary algorithm for solving large-scale multi-modal multi-objective optimization problems, IEEE Trans. Evol. Comput. 25 (3) (2021) 405–418.
[46] Y. Su, Z. Jin, Y. Tian, X. Zhang, K.C. Tan, Comparing the performance of evolutionary algorithms for sparse multi-objective optimization via a comprehensive indicator, IEEE Comput. Intell. Mag. 17 (3) (2022) 34–53.
[47] Y. Tian, X. Zhang, C. Wang, Y. Jin, An evolutionary algorithm for large-scale sparse multiobjective optimization problems, IEEE Trans. Evol. Comput. 24 (2) (2020) 380–393.
[48] S. Jiang, J. Zou, S. Yang, X. Yao, Evolutionary dynamic multi-objective optimisation: A survey, ACM Computing Surveys.
[49] K.K. Bali, A. Gupta, Y.S. Ong, P.S. Tan, Cognizant multitasking in multiobjective multifactorial evolution: MO-MFEA-II, IEEE Trans. Cybern. 51 (4) (2021) 1784–1796.
[50] P. Huang, Y. Wang, A framework for scalable bilevel optimization: Identifying and utilizing the interactions between upper-level and lower-level variables, IEEE Trans. Evol. Comput. 24 (6) (2020) 1150–1163.
[51] Y. Jin, J. Branke, Evolutionary optimization in uncertain environments–a survey, IEEE Trans. Evol. Comput. 9 (3) (2005) 303–317.
[52] W. Du, W. Song, Y. Tang, Y. Jin, F. Qian, Searching for robustness intervals in evolutionary robust optimization, IEEE Trans. Evol. Comput. 26 (1) (2022) 58–72.
[53] M. Li, X. Yao, Quality evaluation of solution sets in multiobjective optimisation: A survey, ACM Comput. Surv. 52 (2) (2019) 26.
[54] Y. Tian, R. Cheng, X. Zhang, M. Li, Y. Jin, Diversity assessment of multi-objective evolutionary algorithms: Performance metric and benchmark problems, IEEE Comput. Intell. Mag. 14 (3) (2019) 61–74.
[55] C.A.C. Coello, N.C. Cortes, Solving multiobjective optimization problems using an artificial immune system, Genet. Program Evolvable Mach. 6 (2) (2005) 163–190.
[56] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257–271.
[57] P. Rakshit, A. Konar, S. Das, Noisy evolutionary optimization algorithms-a comprehensive survey, Swarm Evol. Comput. 33 (2017) 18–45.
[58] S. Yang, Y. Tian, X. Xiang, S. Peng, X. Zhang, Accelerating evolutionary neural architecture search via multi-fidelity evaluation, IEEE Trans. Cogn. Develop. Syst.
[59] D. Gong, X. Ji, J. Sun, X. Sun, Interactive evolutionary algorithms with decision-maker's preferences for solving interval multi-objective optimization problems, Neurocomputing 137 (2014) 241–251.

Ye Tian received the B.Sc., M.Sc., and Ph.D. degrees from Anhui University, Hefei, China, in 2012, 2015, and 2018, respectively. He is currently an Associate Professor with the Institutes of Physical Science and Information Technology, Anhui University, Hefei, China. His current research interests include evolutionary computation and its applications.

Weijian Zhu received the B.Sc. degree from Hefei Normal University, Hefei, China, in 2021, where he is currently pursuing the M.Sc. degree with the School of Computer Science and Technology, Anhui University, Hefei, China. His current research interests include evolutionary computation and its applications.

Xingyi Zhang received the B.Sc. degree from Fuyang Normal College, Fuyang, China, in 2003, and the M.Sc. and Ph.D. degrees from Huazhong University of Science and Technology, Wuhan, China, in 2006 and 2009, respectively. He is currently a Professor with the School of Artificial Intelligence, Anhui University, Hefei, China. His current research interests include unconventional models and algorithms of computation, evolutionary multi-objective optimization, and logistic scheduling.

Yaochu Jin received the B.Sc., M.Sc., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree from Ruhr University Bochum, Germany, in 2001. He is an Alexander von Humboldt Professor for Artificial Intelligence, Chair of Nature Inspired Computing and Engineering, Faculty of Technology, Bielefeld University, Germany. He is also a Distinguished Chair, Professor in Computational Intelligence, Department of Computer Science, University of Surrey, Guildford, U.K. He was a "Finland Distinguished Professor" of University of Jyväskylä, Finland, "Changjiang Distinguished Visiting Professor", Northeastern University, China, and "Distinguished Visiting Scholar", University of Technology Sydney, Australia. His main research interests include evolutionary optimization, evolutionary learning, trustworthy machine learning, and evolutionary developmental systems.