PlatEMO

Evolutionary Multi-Objective Optimization Platform

User Manual

Ye Tian

April 10, 2017

Thank you very much for selecting PlatEMO. The copyright of PlatEMO belongs to the BIMK Group, and 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," 2016.

If you have any comment or suggestion to PlatEMO or the MOEAs in PlatEMO, please send it to *field910921@gmail.com* (*Ye Tian*). If you want to add your MOEA or MOP to PlatEMO, please send the MATLAB code and the relevant literature to *field910921@gmail.com* as well.

We sincerely hope this platform is helpful to your research in evolutionary multi-objective optimization. Your support is the greatest impetus for us!

Contents

I. Introduction to PlatEMO

A. Evolutionary Multi-Objective Optimization

Multi-objective optimization problems (MOPs), which involve two or more conflicting objectives to be optimized, can be formulated as follows:

$$\begin{cases} \min\limits_{X} & F(X) = \left(f_1(X), f_2(X), \dots, f_M(X)\right)^T \\ s.t. & g_i(X) \le 0, \ \ i = 1, \dots, p \end{cases}$$

Where $X = (x_1, x_2, \dots, x_D)^T \in \Omega \subset \mathbb{R}^D$ is the *decision vector* (i.e. *solution*), and Ω is the known *decision space*. $F(X) \in \Lambda \subset \mathbb{R}^M$ is the *objective vector*, and Λ is the unknown *objective space*. $g_i(X)$ includes p inequality *constraints*. Each element in X denotes a *decision variable*, and D is the number of decision variables. Each element in $F(X)$ denotes a single-objective optimization problem, and M is the number of objectives.

Solution X is said to *dominate* solution Y (denoted by $X \prec Y$) if and only if

$$\begin{cases} f_i(X) \le f_i(Y), & \forall i = 1,2, \dots, M \\ f_j(X) < f_j(Y), & \exists j = 1,2, \dots, M \end{cases}$$

And solution X is said to be *Pareto optimal* if and only if

$$\nexists Y \in \Omega: Y \prec X$$

All the Pareto optimal solutions in Ω constitute the *Pareto optimal set*, and the objective values of all the solutions in Pareto optimal set constitute the *Pareto front*.

Many metaheuristics have been verified in solving MOPs in the last two decades, including genetic algorithm, differential evolution, particle swarm optimization, memetic algorithm, estimation of distribution algorithm and so on, which are collectively known as multi-objective evolutionary algorithms (MOEAs). An MOEA usually maintains a *population* consisting of a set of *individuals*, where an individual represents a solution together with its objective values and constraint values. The

population is updated in each generation during the evolution, where new individuals are generated by *operators* (e.g. *crossover* and *mutation*), and the population together with new individuals is truncated by *environmental selection*. The goal of MOEAs is to make the population approximate the Pareto optimal set with good convergence and diversity.

B. PlatEMO

PlatEMO is an open source and free MATLAB-based platform for evolutionary multi-objective optimization, which is available at http://bimk.ahu.edu.cn/index.php?s=/Index/Software/index.html. It can be run on any operating system able to run MATLAB®. PlatEMO provides two running modes to users: command mode and GUI mode. In command mode, no GUI is displayed, users should set the parameters and execute the algorithms by commands. In GUI mode, a GUI is displayed, users can set the parameters and execute the algorithms on the GUI. In order to successfully run the GUI mode, the version of the MATLAB® software should not be lower than R2014b.

The main features of PlatEMO are:

- It includes more than 60 existing MOEAs, most of which are representative algorithms published in top journals after 2010.
- It includes more than 100 popular MOPs, which provide a variety of difficulties for testing the MOEAs.
- It includes a lot of operators for different encodings, which can be combined with all the MOEAs.
- It includes many performance indicators for numerically assessing the performance of MOEAs.
- It is easy to be used that uses need not establish any project or write any code to run PlatEMO, but just invoke the interface function *main*().
- It provides a powerful experimental module in the GUI, which can help users perform experiments on multiple MOEAs and MOPs, and obtain the statistical results in the format of Excel or LaTeX directly.

C. File structure of PlatEMO

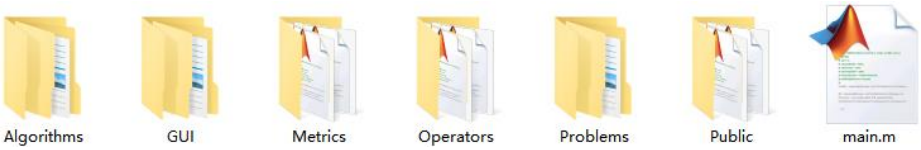

Fig. 1 The file structure of PlatEMO

As shown in Fig. 1, there are six folders and one *.m* file in the root directory of PlatEMO, the functions of which are given in the following.

- *main.m*. The only interface of PlatEMO, invoke this function to run the platform.
- *Algorithms.* For storing the source codes of all the MOEAs.
- *GUI*. For storing the codes to establish the GUI of PlatEMO, which need not be read or modified by users.
- *Metrics*. For storing the source codes of all the performance indicators.
- *Operators*. For storing the source codes of all the operators.
- *Problems*. For storing the source codes of all the MOPs.
- *Public*. For storing some public classes and functions.

All the files in these folders are *.m* files, each of which represents one MATLAB function or MATLAB class with detailed comments. All the files are open source except for the ones in the folder *GUI*.

II. How to Use PlatEMO

A. Use PlatEMO without GUI

Users can run the command mode of PlatEMO by invoking the interface function *main*() with input parameters. While if *main*() is invoked without any input parameter, the GUI mode will be run. All the acceptable parameters for *main*() are listed in Table 1. Note that users need not assign all the parameters since each of them has a default value.

Table 1 The acceptable parameters for *main*()

Parameter Name	Data Type	Default Value	Description
-algorithm	function handle	@NSGAII	MOEA function
-problem	function handle	@DTLZ2	MOP function
-operator	function handle	@EAreal	Operator function
-N	positive integer	100	Population size
-M	positive integer	3	Number of objectives
-D	positive integer	12	Number of variables
-evaluation	positive integer	10000	Number of evaluations
-mode	1, 2 or 3	1	Run mode
-run	positive integer	1	Run No.
-outputFcn	function handle	@GLOBAL.show	Function invoked after each generation
-X_parameter	cell	N/A	Parameters for function X

- *-algorithm*. The function handle of the MOEA to be executed.
- *-problem*. The function handle of the MOP to be solved by the MOEA.
- *-operator*. The function handle of the operator to be used in the MOEA. Note that some MOEAs specify the operator by themselves (e.g. *MOPSO.m*), hence this parameter is invalid for these MOEAs.
- *-N*. The size of the population generated by the MOEA after termination. Note that the population size is fixed to some special values in some MOEAs (e.g. *MOEAD.m*), hence the actual size of the population generated by these MOEAs may not exactly equal to this parameter.

- -*M*. The number of objectives of the MOP. Note that the number of objectives is constant in unscalable MOPs (e.g. *ZDT1.m*), hence this parameter is invalid for these MOPs.
- -*D*. The number of decision variables of the MOP. Note that the number of decision variables is constant or fixed to some integers in some MOPs (e.g. *ZDT5.m*), hence the actual number of decision variables may not exactly equal to this parameter.
- -*evaluation*. The maximum number of fitness evaluations.
- -*mode*. If this parameter is set to 1, a figure showing the result will be displayed after termination. And if this parameter is set to 2, the result will be saved to a file after termination and no figure will be displayed, where the filename is *Data\algorithm\algorithm_problem_M_run.mat* with *algorithm*, *problem*, *M*, and *run* denoting the MOEA name, the MOP name, the number of objectives, and the run number, respectively. If this parameter is set to 3, the result will be neither displayed nor saved, instead, the operation acting on the result is determined by the function *outputFcn*.
- -*run*. The run number. If users want to record multiple results for the same parameters of *algorithm*, *problem* and *M*, modify this parameter in each run so that the filenames of the results are different.
- -*outputFcn*. The function invoked after each generation. In particular, when *mode* is set to 1, this parameter will be forced to @GLOBAL.show, which can display the result after termination; when *mode* is set to 2, this parameter will be forced to @GLOBAL.save, which can save the result after termination.
- -*X_parameter*. For setting the specific parameters in each MOEA, MOP or operator. For example, the algorithm *LMEA.m* has three parameters *nSel*, *nPer* and *nCor*, using *main*(…, '-*LMEA_parameter*', {5,50,5}, …) to set the values of the three parameters to 5, 50 and 5, respectively. If the command is *main*(…, '-*LMEA_parameter*', {[],50}, …), the first parameter *nSel* and the third parameter *nCor* will equal to their default values. Note that the parameter values should be put in a cell array like {5,50,5}, but not an array like [5,50,5]. The specific parameters for each MOEA, MOP and operator can be found in the comments in the head of the corresponding function.

For example, use the following command to run KnEA on DTLZ2 with a population size of 200 and 10 objectives, and the final result will be displayed:

```
1.  main('-algorithm', @KnEA, '-problem', @DTLZ2, '-N', 200, '-M', 10);
```

Use the following command to run RVEA on UF8 with DE operator for 10 times, the parameters in *DE.m*, i.e. *CR*, *F*, *proM* and *disM*, are set to 1, 0.6, 1 and 10, respectively, and the final results will be saved:

```
for r = 1 : 10
    main('-algorithm', @RVEA, '-problem', @UF8, '-mode', 2, '-run', r, ...
    '-DE_parameter', {1,0.6,1,10});
end
```

B. Use PlatEMO with GUI

Users can run the GUI mode of PlatEMO by the following command:

```
main();
```

Then two modules can be seen on the GUI, i.e. the test module and the experimental module. The test module is used to execute one MOEA on an MOP each time, and the result will be displayed in a figure. The experimental module is used to execute multiple MOEAs on several MOPs at the same time, and the statistical results will be listed in a table and saved to files.

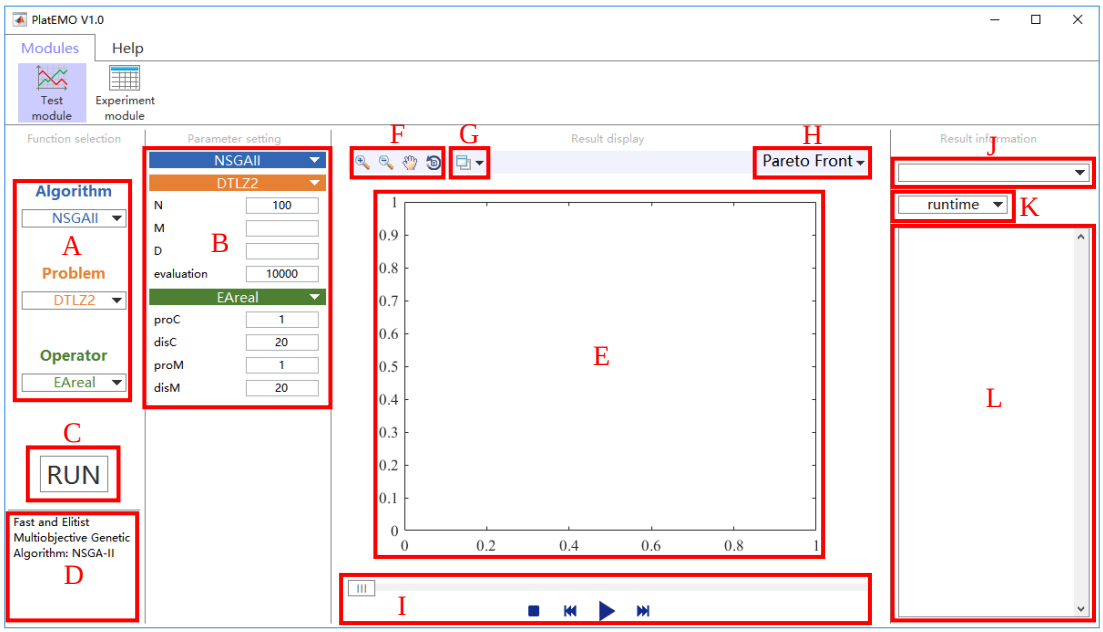

Fig. 2 The interface of the test module

The interface of the test module of PlatEMO is shown in Fig. 2. The functions of the

controls in each region are:

- *Region A*. Select the MOEA, MOP and operator to be executed. Note that some MOEAs specify the operator by themselves (e.g. *MOPSO.m*), hence the operator cannot be selected by users when selecting these MOEAs.
- *Region B*. Set the parameters of the selected MOEA, MOP and operator. The value of each parameter should be a scalar. Note that here the common parameters *N*, *M*, *D* and *evaluation* are regarded as the parameters of MOPs. A parameter will equal to its default value if the value assigned by users is empty.
- *Region C*. Execute the algorithm according to the current configuration.
- *Region D*. Show the introduction of the parameter in *Region B* which the cursor is moving over.
- *Region E*. Show the current population during the optimization.
- *Region F*. Zoom in, zoom out, pan or rotate the axis in *Region E*.
- *Region G*. Open the axis in *Region E* in a new standard MATLAB figure, thus more operations can be acted on the axis, e.g. saving the axis.
- *Region H*. Select the data to be displayed in the axis in *Region E*, including the Pareto front of the population, the Pareto set of the population, the true Pareto front of the MOP, and the evolutionary trajectory of any performance indicator.
- *Region I*. Control the optimization procedure, e.g. start, pause, stop, backward and forward.
- *Region J*. Select one of the historical results to be displayed.
- *Region K*. Show the value of any performance indicator of the final population of the result.
- *Region L*. Show the detailed information of the result.

After open the text module, users should first select the MOEA, MOP and operator to be executed in *Region A*, and set their parameters in *Region B*, then press the button in *Region C* to execute the algorithm. The real-time population will be displayed in the axis in *Region E*, and users can use the buttons in *Region I* to control the optimization procedure. After the algorithm has been terminated, all the historical results can be redisplayed by selecting the popup menu in *Region J*.

The interface of the experimental module of PlatEMO is shown in Fig. 3. The functions of the controls in each region are:

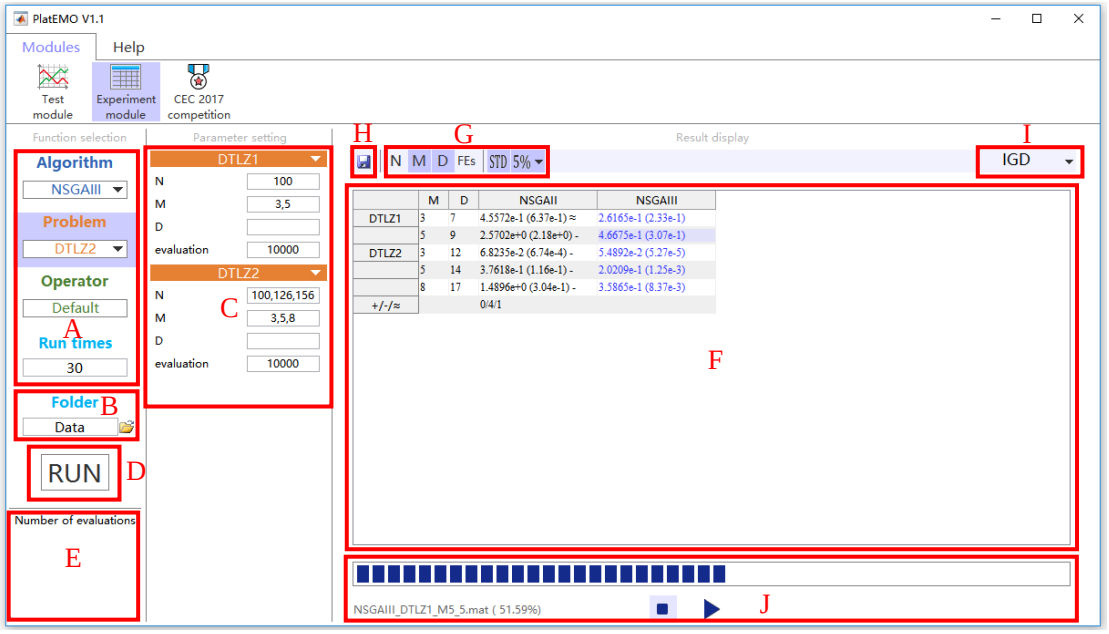

Fig. 3 The interface of the experimental module

- *Region A*. Select the MOEAs and MOPs to be executed and specify the run times. Note that multiple MOEAs and MOPs can be selected here, but the operators cannot be selected since only the default operators can be used.
- *Region B*. Set the folder for saving the results. Uses can also open an existing configuration file so that all the MOEAs and MOPs to be executed and their parameter values can be automatically set. The configuration file *Setting.mat* will be automatically created in the specified folder when starting to execute the experiment.
- *Region C*. Set the parameters of the selected MOEAs, MOPs and operators. Note that the value of each parameter in the MOPs can be a vector, thus the MOEAs can be executed on the same MOP with different settings at the same time. Note that the length of the vector should be 0, 1 or the same to the number of distinct values of parameter *M*.
- *Region D*. Execute the experiment according to the current configuration.
- *Region E*. Show the introduction of the parameter in *Region B* which the cursor is moving over.
- *Region F*. Show the statistical results of the experiment.
- *Region G*. Specify the type of data to be shown in the table in *Region F*.
- *Region H*. Save the table in *Region F* in the format of Excel or LaTeX.
- *Region I*. Select the data to be shown in the table in *Region F*, including the values of any performance indicator of the final populations of the results.

- *Region J*. Control the optimization procedure.

After opening the experimental module, users should first select the MOEAs and MOPs to be executed in *Region A* and set their parameters in *Region C*, or load existing configuration by pressing the button in *Region B*, then press the button in *Region D* to execute the experiment. The statistical results will be displayed in the table in *Region F*, and users can use the buttons in *Region J* to control the optimization procedure. After the experiment has been terminated, the data shown in the table in *Region F* can be saved in the format of Excel or LaTeX by pressing the button in *Region H*.

III. How to Extend PlatEMO

A. Architecture of PlatEMO

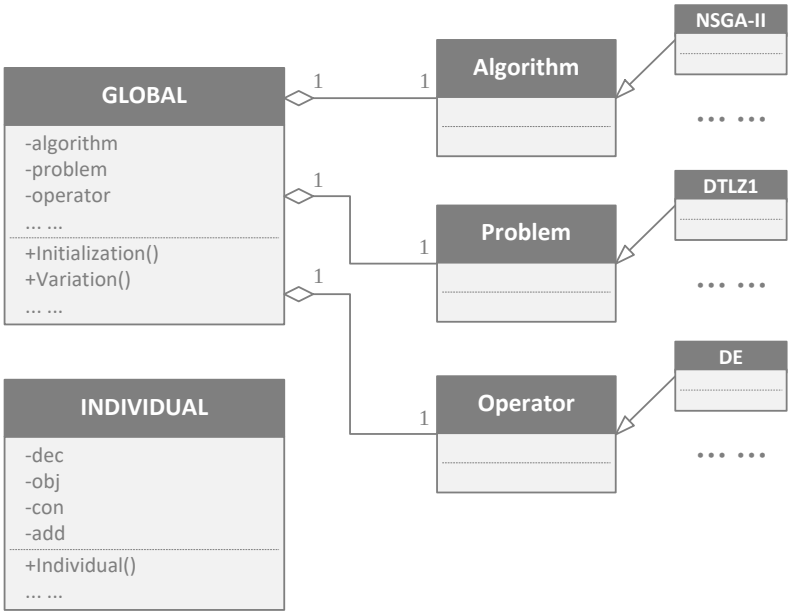

Fig. 4 Architecture of PlatEMO

The architecture of PlatEMO is shown in Fig. 4, where there are only two classes in the implementation of PlatEMO, i.e. *GLOBAL* and *INDIVIDUAL*. Note that each algorithm, problem and operator is just a function, but not an object.

GLOBAL represents the configuration of the current run, the source code and the detailed comments of which can be found in *Public\GLOBAL.m*. During each execution, one *GLOBAL* object is maintained to store all the parameter setting and the result data, including the function handles of the executed MOEA, MOP and operator, the population size, the number of objectives, the number of decision variables, the maximum number of fitness evaluations, the number of evaluated fitness, and so on. *GLOBAL* also provides some methods which can be invoked by the MOEAs, for instance, *GLOBAL.Initialization*() can generate a randomly initial population, and *GLOBAL.Variation*() can generate a set of offsprings according to specified parents.

INDIVIDUAL represents one individual, the source code and the detailed comments of which can be found in *Public\INDIVIDUAL.m*. An *INDIVIDUAL* object stores the decision variables *dec*, the objective values *obj*, the constraint values *con*, and the additional property values *add* of one individual. The values of *dec* and *add* are assigned

when invoking the constructor, then the values of *obj* and *con* are calculated automatically. Each of the above properties is a row vector, and use *P.decs*, *P.objs*, *P.cons* or *P.adds* can obtain a matrix of the decision variables, objective values, constraint values or additional property values of an array of *INDIVIDUAL* objects *P*, respectively, where each row of the matrix denotes one individual and each column denotes one dimension of the values. When the number of instantiated *INDIVIDUAL* objects exceeds the maximum number of fitness evaluations, the algorithm will be forced to be terminated.

B. Add Algorithms

An MOEA function is represented by an *.m* file in PlatEMO, which should be put in the folder *Algorithms*. For example, the source code of *NSGAII.m* is

```
1.  function NSGAII(Global)
2.      Population = Global.Initialization();
3.      FrontNo = NDSort(Population.objs, inf);
4.      CrowdDis = CrowdingDistance(Population.objs, FrontNo);
5.      while Global.NotTermination(Population)
6.          MatingPool = TournamentSelection(2, Global.N, FrontNo, -CrowdDis);
7.          Offspring = Global.Variation(Population(MatingPool));
8.          [Population, FrontNo, CrowdDis] = ...
9.          EnvironmentalSelection([Population, Offspring], Global.N);
10.     end
11. end
```

Note that the common codes for all the MOEAs are underlined. To begin with, an MOEA function has one input parameter denoting the *GLOBAL* object and none output parameter. Then an initial population *Population* is generated (line 2), and the non-dominated front number and the crowding distance of each individual are calculated (line 3-4). In each generation, *Global.NotTermination*() is invoked to check whether the number of evaluated fitness exceeds the maximum number of fitness evaluation, and *Population* is passed to the function to be the final output (line 5). Afterwards, the mating pool selection, generating offsprings, and environmental selection are performed in sequence (line 6-9).

As shown in the code of NSGA-II, one MOEA should perform three operations at least: obtaining an initial population via *Global.Initialization*(), checking the

optimization state and passing *Population* via *Global.NotTermination*(), and generating offsprings via *Global.Variation*(). Besides, the function *CrowdingDistance*() and *EnvironmentalSelection*() are specific to NSGA-II, and *NDSort*() and *TournamentSelection*() are public functions stored in the folder *Public*.

For decomposition based MOEAs, a set of reference points should be generated before the optimization. For example in *MOEAD.m*, it uses the following command to generate the reference points:

```
1.  [W, Global.N] = UniformPoint(Global.N, Global.M);
```

Where *UniformPoint*() is a public function in the folder *Public* for generating about *Global.N* uniformly distributed points with *Global.M* objectives on the unit hyperplane. *W* is the set of reference points, and the population size *Global.N* is reset to the same to the number of reference points in *W*.

The comments in the head of the MOEA functions (as well as MOP, operator and performance indicator functions) should be written in a specified form such that it can be identified by PlatEMO. For example in *MOEADDE.m*, the comments in the head of the function are

```
1.  function MOEADDE(Global)
2.  % <algorithm> <H-N>
3.  % Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D
4.  % and NSGA-II
5.  % delta --- 0.9 --- The probability of choosing parents locally
6.  % nr --- 2 --- Maximum number of solutions replaced by each offspring
7.  % operator --- DE
```

Line 2 gives two labels of the function, the first label <algorithm> indicates that this is an MOEA function (it is <problem>, <operator> and <metric> for MOP, operator and performance indicator function, respectively), and the second label <H-N> can be an arbitrary string. Line 3-4 give the title of the relevant literature. Line 5-6 define the parameters for MOEA/D-DE, where the name of the parameter is located on the first column of each line, the default value of the parameter is located on the second column, and the introduction to the parameter is located on the third column; the columns are divided by --- . Line 7 specifies the operator for MOEA/D-DE, where this line is optional.

MOEA/D-DE then receives the parameter setting from users by the following

command:

```
[delta, nr] = Global.ParameterSet(0.9, 2);
```

The detailed introduction to *Global.ParameterSet*() can be found in *Public\GLOBAL.m*.

An MOEA may use its specified operator to generate offsprings, for example in *MOEADDE.m*, the following command is used to generate offsprings by *DE.m*:

```
Offspring = Global.Variation(Population(MatingPool), 1, @DE);
```

The detailed introduction to *Global.Variation*() can be found in *Public\GLOBAL.m*. Note that the number of evaluated fitness will be increased once this function is invoked.

For surrogate-assisted MOEAs, the following command can be used to generate new decision variables according to the decision variables of parents, while no *INDIVIDUAL* object will be instantiated:

```
OffDec = Global.VariationDec(Population(MatingPool).decs);
```

The detailed introduction to *Global.VariationDec*() can be found in *Public\GLOBAL.m*.

C. Add Problems

An MOP function is represented by an *.m* file in PlatEMO, which should be put in the folder *Problems*. For example, the source code of *DTLZ2.m* is

```
function varargout = DTLZ2(Operation, Global, input)
    switch Operation
        case 'init'
            Global.M = 3;
            Global.D = Global.M + 9;
            Global.lower = zeros(1, Global.D);
            Global.upper = ones(1, Global.D);
            Global.operator = @EAreal;
            PopDec = rand(input, Global.D);
            varargout = {PopDec};
        case 'value'
            PopDec = input;
            M = Global.M;
            PopObj = zeros(size(PopDec, 1), M);
```

```
15.                 g = sum((PopDec(:, M:end)-0.5).^2, 2);
16.                 for m = 1 : M
17.                     PopObj(:, m) = (1+g).*prod(cos(PopDec(:, 1:M-m)*pi/2), 2);
18.                     if m > 1
19.                         PopObj(:, m) = PopObj(:, m).*sin(PopDec(:, M-m+1)*pi/2);
20.                     end
21.                 end
22.                 PopCon = [];
23.                 varargout = {PopDec, PopObj, PopCon};
24.             case 'PF'
25.                 f = UniformPoint(input, Global.M);
26.                 f = f./repmat(sqrt(sum(f.^2, 2)), 1, Global.M);
27.                 varargout = {f};
28.         end
29. end
```

Note that the common codes for all the MOPs are underlined. To begin with, an MOP function has three input parameters and one output parameter, where *Operation* denotes the operation to be done, *Global* denotes the *GLOBAL* object, and *input* and *varargout* may have one of the three meanings when *Operation* is set to different values. To be specific, if *Operation* is set to 'init', the MOP function will assign the default values of *Global.M*, *Global.D*, *Global.lower*, *Global.upper* and *Global.operator* (line 4-8), and return the decision variable matrix of a random population with size *input* (line 9-10). If *Operation* is set to 'value', the parameter *input* will denote the decision variable matrix of a population, and the objective values *PopObj* and constraint values *PopCon* of the population are calculated (line 14-22), note that here *PopCon* is set to empty since DTLZ2 does not have any constraint. Then the matrices of decision variables, objective values and constraint values of the population are returned (line 23). If *Operation* is set to 'PF', *input* uniformly distributed reference points will be sampled on the true Pareto front of the MOP and returned (line 25-27). Note that if the true Pareto front of the MOP is unknown, the MOP should also return a nadir point for the calculation of hypervolume (e.g. *MOKP.m*).

The number of objectives and decision variables may be fixed to some special values in some MOPs. For example in *ZDT5.m*, these two values are assigned by

```
1.  Global.M = 2;
2.  Global.M = 2;
3.  Global.D = 80;
4.  Global.D = ceil(max(Global.D-30,1)/5)*5 + 30;
```

In line 1, the function sets the default value of *Global.M* to 2; if *Global.M* has already been assigned by users, this line of code will be invalid. In line 2, the function sets the value of *Global.M* to 2 again, then *Global.M* will equal to 2 no matter whether users have assigned it. Analogously, line 3 sets the default value of *Global.D* to 80, and line 4 forces the value of *Global.D* to be a multiple of 5 after being subtracted by 30.

It is worth to note that a decision variable is illegal in three cases. First, for continuous MOPs, it may be greater than the upper bound *Global.upper* or less than the lower bound *Global.lower*, in this case it will be set to the boundary value by *INDIVIDUAL* class, so MOP functions need not handle this case. Second, for combinational MOPs, it may be an illegal character, in this case it should be modified and returned by the MOP function when calculating the objective values and constraint values (e.g. *MOKP.m*). Thirdly, it may not fulfill the constraints (a positive constraint value indicates that this constraint is not fulfilled), while in this case it will not be modified.

An MOP function can receive parameter setting similar to MOEA functions, for example in *WFG1.m*, the following command is used to receive parameter setting from user:

```
1.  K = Global.ParameterSet(Global.M-1);
```

While in some combinatorial MOP functions, the parameter is a random matrix and should be identical in different runs, therefore such parameters should be randomly generated and then saved to files. For example in *MOKP.m*, the following command is used to set such parameters:

```
1.  [P, W] = Global.ParameterFile(springf('MOKP-M%d-D%d', M, D), ...
2.                                randi([10, 100], M, D), randi([10, 100], M, D));
```

To be specific, if the specified data file does not exist, the two parameters will be set to the random matrices, and the matrices will be saved to the specified data file; while if the file exists, the two parameters will be set to the matrices saved in the file. The detailed introduction to *Global.ParameterFile*() can be found in *Public\GLOBAL.m*.

D. Add Operators

An operator function is represented by an *.m* file in PlatEMO, which should be put in the folder *Operators*. For example, the source code of *EAbinary.m* is

```
1.  function Offspring = EAbinary(Global, Parent)
2.      Parent = Parent(min(1:ceil(end/2)*2, end));
3.      parentDec = Parent.decs;
4.      [N,D] = size(parentDec);
5.      Parent1Dec = parentDec(1:N/2, :);
6.      Parent2Dec = parentDec(N/2+1:end, :);
7.      k = repmata(1:D, N/2, 1) > repmat(randi(D, N/2, 1), 1, D);
8.      Offspring1Dec = Parent1Dec;
9.      Offspring2Dec = Parent2Dec;
10.     Offspring1Dec(k) = Parent2Dec(k);
11.     Offspring2Dec(k) = Parent1Dec(k);
12.     OffspringDec = [Offspring1Dec; Offspring2Dec];
13.     site = rand(N, D) < 1/D;
14.     OffspringDec(site) = ~OffspringDec(site);
15.     Offspring = INDIVIDUAL(OffspringDec);
16. end
```

Note that the common codes for all the operators are underlined. To begin with, an operator function has two input parameters and one output parameter, where *Global* denotes the *GLOBAL* object, *Parent* denotes the parent population, and *Offspring* denotes the new population generated based on *Parent*. Since the number of parents for crossover should be a multiple of two, the operator function first makes the number of parents be an even (line 2). Then, it calculates the decision variables of the offsprings according to the decision variables of the parents, where the single-point crossover and bitwise mutation are performed in line 7-12 and line 13-14, respectively. Finally, the objects of the offsprings are generated by *INDIVIDUAL* class and returned (line 15).

In particular, an individual may have additional property in some operators. For example in *PSO.m*, it uses the following command to obtain the values of the speed property of all the parents:

```
1.  Speed = Parent.adds(zeros(N, D));
```

Where the input parameter *zeros*(*N*, *D*) is the matrix of default values for the speed property of all the parents, that is, set the speed property of a parent to the default value if it does not include this property. Then, the function uses the following command to assign the speed property of offsprings:

```
1.  Offspring = INDIVIDUAL(NewDec, NewSpeed);
```

Where *NewDec* is the matrix of decision variables of all the offsprings, and *NewSpeed* is the matrix of speed property of all the offsprings. Note that an individual can have more than one additional properties, while the operator functions need not specify the name of each additional property, because the name of an additional property is automatically set to the same to the name of the operator function.

E. Add Performance Indicators

A performance indicator function is represented by an *.m* file in PlatEMO, which should be put in the folder *Metrics*. For example, the source code of *IGD.m* is

```
1.  function score = IGD(PopObj, PF)
2.      score = mean(min(pdist2(PF, PopObj), [], 2);
3.  end
```

Note that the common codes for all the performance indicators are underlined. To begin with, a performance indicator function has two input parameters and one output parameter, where *PopObj* denotes the matrix of objective values of the population, *PF* denotes the set of reference points sampled on the true Pareto front of the MOP, and *score* denotes the scalar performance indicator value. Then, the performance indicator value of the population is calculated according to the reference points and returned (line 2).