PlatEMO

Evolutionary Multi-Objective Optimization Platform

User Manual 2.0

BIMK Group

December 15, 2018

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

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

If you have any comment or suggestion to PlatEMO 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. You can obtain the newest version of PlatEMO from https://github.com/BIMK/PlatEMO.

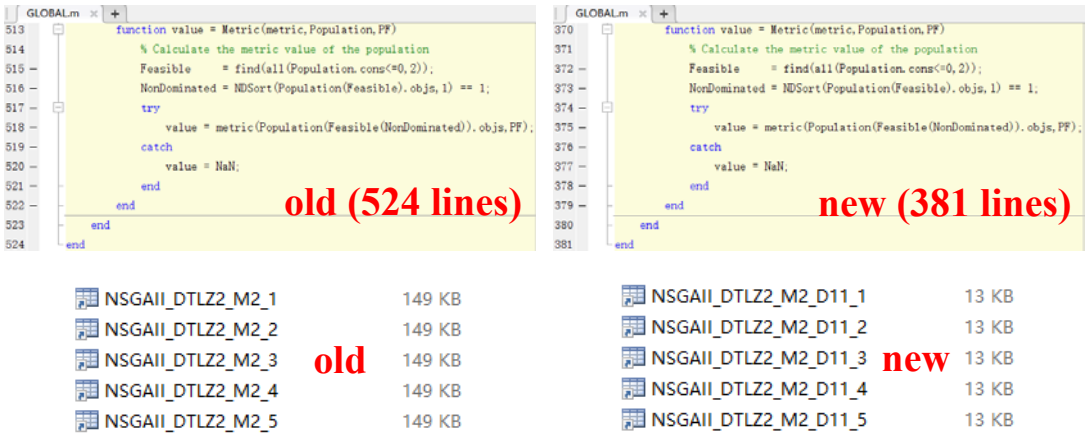
Contents

I. Release Notes

We strongly recommend the users who have used PlatEMO v1.0-v1.6 to read this section before using PlatEMO 2.0.

A. Release Highlights

- **Lighter framework.** The architecture of PlatEMO is simplified, which leads to lower learning cost and higher efficiency. The result file size is also reduced.



- **Higher efficiency.** The runtime of Pareto dominance based algorithms is reduced by using a more efficient non-dominated sorting algorithm. The runtime of decomposition based algorithms is reduced due to the new architecture of PlatEMO. The runtime of hypervolume calculation is reduced by new logic and GPU acceleration. In experimental module, the algorithms can be executed in parallel.

```
NSGAII on DTLZ2, 10 objectives, run 1 (100.46%), 2.26s passed...
>>
```
old

```
NSGAII on DTLZ2, 10 objectives 19 variables, run 1 (100.29%), 1.71s passed...
>>
```
new

```
MOEAD on DTLZ2, 3 objectives, run 1 (100.28%), 7.34s passed...
>>
```
old

```
MOEAD on DTLZ2, 3 objectives 12 variables, run 1 (100.28%), 4.83s passed...
>>
```
new

```
>> tic; HV(UniformPoint(275,10),ones(1,10)); toc
Elapsed time is 16.707720 seconds.
```
old

```
>> tic; HV(UniformPoint(275,10),ones(1,10)); toc
Elapsed time is 3.952227 seconds.
```
new

- **More conveniences.** The populations obtained during the evolutionary process can be saved in result files. The references of each algorithm, problem, operator, and metric are given in the comments of the function. The codes of GUI are now open source.

```
Population =                    result =
  1×105 INDIVIDUAL array          5×2 cell array
                                    {[ 3953]}    {1×105 INDIVIDUAL}
    dec                             {[ 7905]}    {1×105 INDIVIDUAL}
    obj            old              {[11961]}    {1×105 INDIVIDUAL}  new
    con                             {[15913]}    {1×105 INDIVIDUAL}
    add                             {[20073]}    {1×105 INDIVIDUAL}
```

B. New Features

- **The operator cannot be specified by users anymore.** Algorithms should directly invoke an operator function (e.g., `GA()`, `DE()`, and `PSO()`) to generate offsprings instead of `GLOBAL.Variation()`. Problems should specify the encoding `GLOBAL.encoding` (e.g.,`'real'`, `'binary'`, and `'permutation'`) instead of the operator function `GLOBAL.operator`.
- **All the problem functions are converted into classes.** Each problem should be written as a class and inherit the class `PROBLEM`, and each operation (e.g., initialization, calculation of objective values, and calculation of constraint violations) is an overloaded method.
- **The populations obtained during the evolutionary process can be saved.** Each result file contains two variables `result` and `metric`, where `result` is a cell with the first column storing the number of evaluations at each time point and the second column storing the population obtained at each time point. Use `main(…,'-save',N,…)` to set the number of saved populations to `N`. The reference point set `PF` is no longer stored in the result file.
- **The algorithms can be executed in parallel in experimental module.** The number of workers depends on the number of CPUs of the machine.
- **The algorithms can be executed on a problem with different numbers of objectives or decision variables in experimental module**. It allows either the

number of objectives or the number of decision variables or both of them to be vectors. The filename of a result file is like `NSGAII_DTLZ2_M2_D11_1.mat`.

- **The convergence profile of any metric values can be shown in experimental module.** The convergence profile can be shown by right-clicking on a cell in the table in experimental module. The number of metric values in the figure depends on the number of saved populations, which can also be set in experimental module.
- **The code of hypervolume is enhanced and accelerated.** The function `HV()` in the new version is the same to `NHV()` in the old version, and the function `NHV()` is removed. The efficiency of hypervolume calculation is highly improved by new logic and GPU acceleration.
- **The code of non-dominated sorting is accelerated.** When the number of individuals and the number of objectives are large, the T-ENS is adopted to perform non-dominated sorting instead of ENS-SS.
- **The way to set the parameters in algorithms and problems is changed.** Use `main(…,'-algorithm',{@MOEAD,2},…)` to set the parameter in MOEA/D instead of `main(…,'-MOEAD_parameter',{2},…)`.
- **Add a public function for performing roulette-wheel selection.** The function `RouletteWheelSelection()` can return a set of individuals generated by roulette-wheel selection.
- **Provide the references of each function.** The references of each algorithm, problem, operator, and metric are given in the comments of the function.
- **The codes of GUI are now open source,** which can be reused by users.

C. Compatibility Considerations

- **The function `GLOBAL.Variation()` or `GLOBAL.VariationDec()` cannot be invoked anymore.** Each algorithm should invoke an operator function like `GA()`, `DE()`, and `PSO()`. Note that the interfaces of different operator functions may be different.
- **Each problem should be written as a subclass of `PROBLEM`**, and each operation (e.g., repairing infeasible variables, calculating objective values, and calculating constraint violations) should be written as an overloaded method, but not a branch of `switch-case-end`.
- **The filename and variables in the result file are modified.** The filename and variables are different to those in the old version, hence the data generated by the old version cannot be loaded in the experimental module of the new version.

II. Introduction to PlatEMO

A. Evolutionary Multi-Objective Optimization

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

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

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*. $g_i(X)$ are 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 an objective function, and M is the number of objectives.

Solution X is said to *Pareto 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 set*, and the objective values of all the solutions in Pareto set constitute the *Pareto front*.

Many metaheuristics have been employed to solve 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 violations. The population is updated in each generation of 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 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 and https://github.com/BIMK/PlatEMO. PlatEMO can be used on any operating system able to run MATLAB®. PlatEMO provides two running modes to users, i.e., command mode and GUI mode. In command mode, no GUI is displayed, and users should set the parameters and execute the algorithms by commands. In GUI mode, a GUI is displayed, and users can set the parameters and execute the algorithms on the GUI. In order to successfully use the GUI mode, the version of MATLAB® software should not be lower than R2014b.

The main features of PlatEMO are:

- It includes more than 90 existing MOEAs, most of which are representative algorithms published in top journals after 2010.
- It includes more than 120 benchmark MOPs, which provide a variety of difficulties for testing the MOEAs.
- It includes many performance metrics for quantitatively measuring the performance of MOEAs.
- Users 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

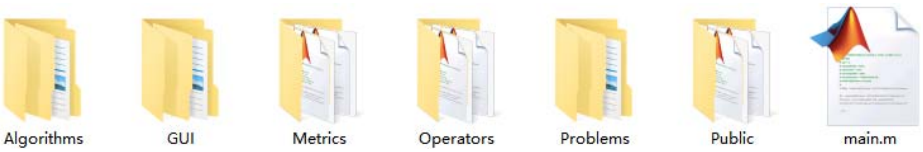


Fig. 1 The file structure of PlatEMO

As shown in Fig. 1, there are six folders and one `.m` file in the root directory of PlatEMO, the functions of which are listed 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.
- `Metrics`. For storing the source codes of all the performance metrics.
- `Operators`. For storing the source codes of all the operators.
- `Problems`. For storing the source codes of all the MOPs.
- `Public`. For storing public classes and utility functions.

All the files in these folders are `.m` files, each of which represents one MATLAB function or MATLAB class. All the files are open source and with detailed comments.

III. 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
`-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
`-run`	positive integer	`1`	Run number
`-save`	integer	`0`	Number of saved populations
`-outputFcn`	function handle	`@GLOBAL.Output`	Function invoked after each generation

- `-algorithm`. The function handle of the MOEA to be executed.
- `-problem`. The function handle of the MOP to be solved.
- `-N`. The population size of the MOEA. Note that it is fixed to some particular values in some MOEAs (e.g. `MOEAD.m`), hence the actual population size of 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 particular 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 function evaluations.
- `-run`. The run number. If users want to save multiple results for the same parameters of `algorithm`, `problem`, `M` and `D`, modify this parameter in each run so that the filenames of the results are different.
- `-save`. The number of saved populations. If this parameter is set to 0 (default), a figure showing the result will be displayed after termination; otherwise, the populations obtained during the evolutionary process will be saved in a file named as `Data\algorithm\algorithm_problem_M_D_run.mat`. For example, if `save` is 5 and `evaluation` is 20000, the populations obtained when the number of evaluation is 4000, 8000, 12000, 16000 and 20000 will be saved.
- `-outputFcn`. The function invoked after each generation, which need not be modified in general.

For example, use the following command to run RVEA on WFG1 with a population size of 200 and 10 objectives, and the final result will be displayed:

```
main('-algorithm',@RVEA,'-problem',@WFG1,'-N',200,'-M',10);
```

Use the following command to run KnEA on WFG2, and set the parameters in KnEA and WFG2:

```
main('-algorithm',{@KnEA,0.4},'-problem',{@WFG2,6});
```

The specific parameters for each MOEA and MOP can be found in the comments in the head of the corresponding function. Use the following command to run AR-MOEA on DTLZ5 for 10 times, and the final population will be saved:

```
for r = 1 : 10
    main('-algorithm',@ARMOEA,'-problem',@DTLZ5,'-save',1,
    '-run',r);
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.

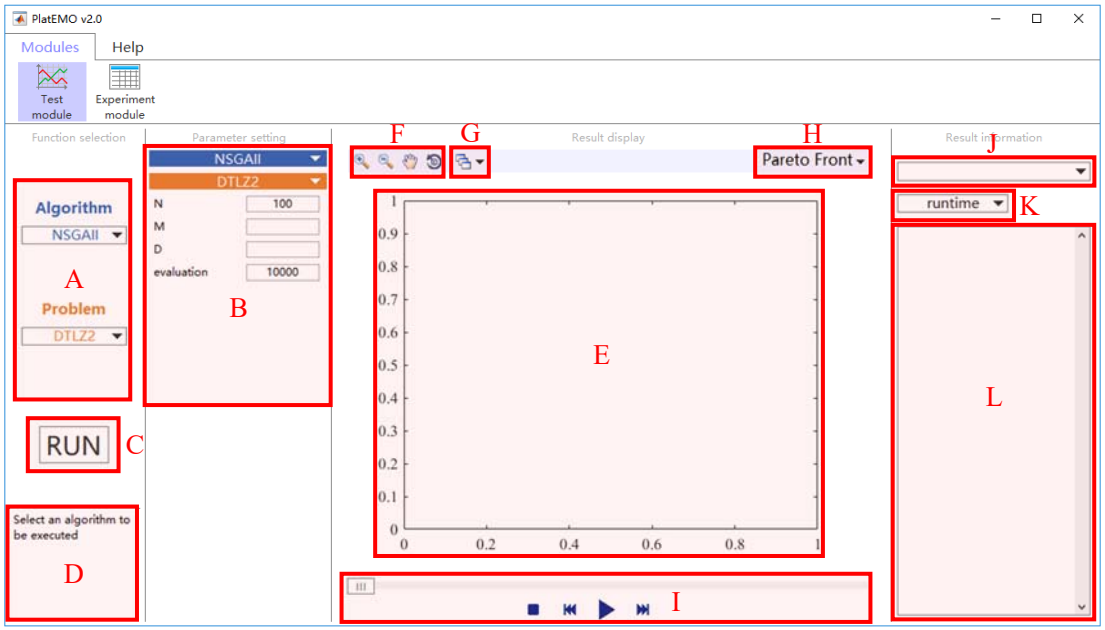


Fig. 2 The interface of test module

The interface of test module of PlatEMO is shown in Fig. 2. The functions of the controls in each region are:

- *Region A*. Select the MOEA and MOP to be executed.
- *Region B*. Set the parameters of the selected MOEA and MOP. 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 the MOP. A parameter will equal to its default value if it is set to empty.
- *Region C*. Execute the MOEA 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 convergence profile of any performance metric.

- *Region I*. Control the optimization procedure, i.e., start, pause, stop, backward and forward.
- *Region J*. Show one of the historical results.
- *Region K*. Show a performance metric value of the final population of the result.
- *Region L*. Show the detailed information of the execution.

After opening the test module, users should first select the MOEA and MOP 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 is terminated, all the historical results can be redisplayed by selecting the popup menu in *Region J*.

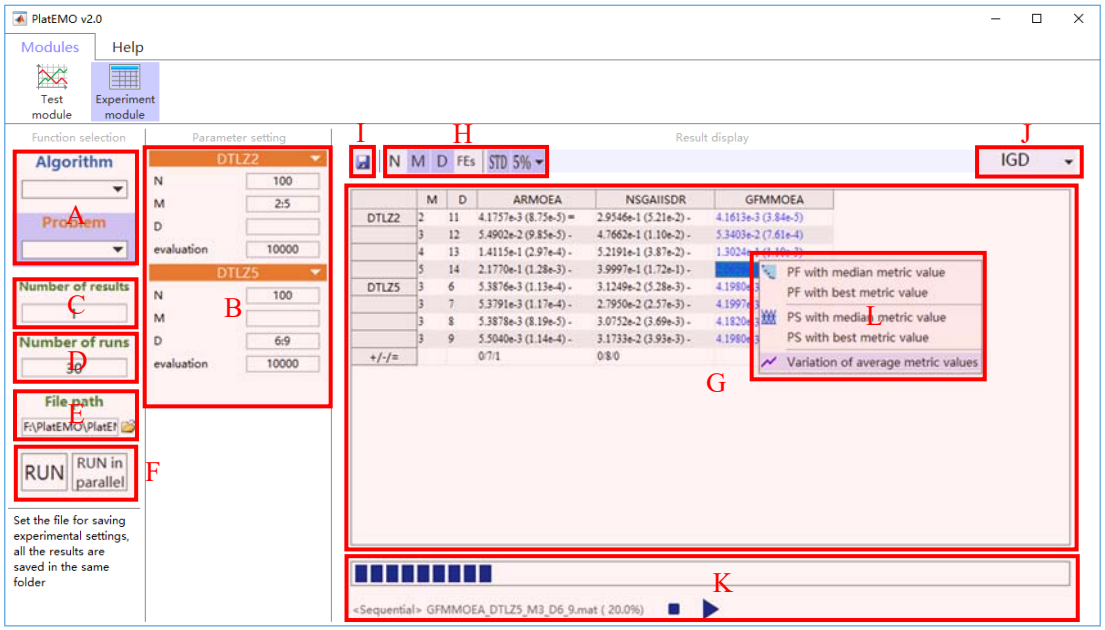


Fig. 3 The interface of experimental module

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

- *Region A*. Select multiple MOEAs and MOPs to be executed.
- *Region B*. Set the parameters of the selected MOEAs and MOPs. 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.

- *Region C*. Set the number of populations saved in each file. For example, if the number of populations is 5 and the number of evaluations is 20000, the populations obtained when the number of evaluation is 4000, 8000, 12000, 16000 and 20000 will be saved.
- *Region D*. Set the number of runs of each MOEA on each MOP.
- *Region E*. Set the file path for saving the experimental settings. Users can also open an existing configuration file to load experimental settings. All the results will be saved in the same folder to the file path.
- *Region F*. Execute the experiment in sequence or in parallel.
- *Region G*. Show the statistical results of the experiment.
- *Region H*. Specify the type of data to be shown in the table.
- *Region I*. Save the table in the format of Excel or LaTeX.
- *Region J*. Select the data to be shown in the table, including any performance metric values of the final populations.
- *Region K*. Control the optimization procedure, i.e., start, pause and stop.
- *Region L*. Right-click on a cell to show the Pareto front of the population, the Pareto set of the population, or the convergence profile of metric values.

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 B*, the number of saved populations in *Region C*, and the number of runs in *Region D*. Then, press one of the two buttons in *Region F* to start the experiment. The statistical results will be displayed in the table in *Region G*, and users can use the buttons in *Region K* to control the optimization procedure. After the experiment is terminated, the data shown in the table can be saved in the format of Excel or LaTeX by pressing the button in *Region I*.

Alternatively, users can load existing configurations by pressing the button in *Region E*, and then start the experiment. If any result file has already existed in the folder given in *Region E*, the result file will be loaded instead of executing the algorithm.

IV. How to Extend PlatEMO

A. Architecture of PlatEMO

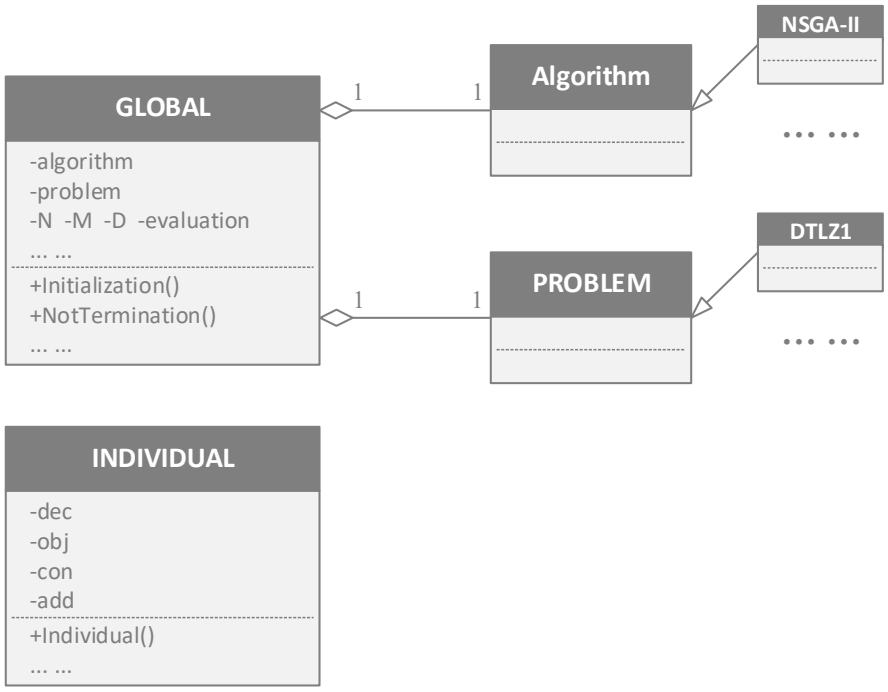


Fig. 4 Architecture of PlatEMO

The architecture of PlatEMO is shown in Fig. 4, where there are three classes in the implementation of PlatEMO, i.e. `GLOBAL`, `INDIVIDUAL` and `PROBLEM`. Note that each algorithm is a function, and 'Algorithm' is not a class but just a unified interface for all the algorithm functions.

`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 settings and the result, including the function handles of the executed MOEA and MOP, the population size, the number of objectives, the number of decision variables, the maximum number of function evaluations, the number of evaluated individuals, and so on. `GLOBAL` also provides some methods which can be invoked by the MOEAs, for instance, `GLOBAL.Initialization()` can generate a random initial population, and `GLOBAL.NotTermination()` can check whether the algorithm should be terminated or not.

`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 violations `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 `P.decs`, `P.objs`, `P.cons` or `P.adds` denotes a matrix of the decision variables, objective values, constraint violations 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. The number of evaluated individuals `GLOBAL.evaluated` will be increased once new `INDIVIDUAL` objects are instantiated, and the algorithm will be forced to be terminated when the number of instantiated `INDIVIDUAL` objects exceeds the maximum number of function evaluations `GLOBAL.evaluation`,

`PROBLEM` is the superclass for all the problem classes. It contains several methods, i.e., `PROBLEM.Init()` for generating an initial population, `PROBLEM.CalDec()` for repairing infeasible decision variables, `PROBLEM.CalObj()` for calculating objective values, `PROBLEM.CalCon()` for calculating constraint violations, and `PROBLEM.PF()` for sampling reference points on the true Pareto front. Each problem should be written as a subclass of `PROBLEM` and overload the above methods.

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,CrowdDis] =
       EnvironmentalSelection(Population,Global.N);
4.     while Global.NotTermination(Population)
5.         MatingPool = TournamentSelection(2,Global.N,FrontNo,
           -CrowdDis);
6.         Offspring = GA(Population(MatingPool));
7.         [Population,FrontNo,CrowdDis] =
           EnvironmentalSelection([Population,Offspring],
           Global.N);
8.     end
9. end
```

To begin with, an MOEA function has one input parameter denoting the `GLOBAL` object

and zero 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). In each generation, `Global.NotTermination()` is invoked to check whether the number of evaluated fitness exceeds the maximum number of function evaluation, and `Population` is passed to the function to be the final output (Line 4). Afterwards, the mating pool selection, generating offsprings, and environmental selection are performed in sequence (Lines 5-7).

As a result, 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 an operator function such as `GA()`. Besides, the function `EnvironmentalSelection()` is specific to NSGA-II, and `NDSort()` and `TournamentSelection()` are utility functions stored in the folder `Public`.

For decomposition based MOEAs, a set of reference points should be generated beforehand. For example in `MOEAD.m`, the following command is used to generate the reference points:

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

Where `UniformPoint()` is a utility 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 functions) should be written in a specified form such that it can be identified by the GUI. For example in `GFMMOEA.m`, the comments in the head of the function are

```
1.  function GFMMOEA(Global)
2.  % <algorithm> <G>
3.  % Generic front modeling based MOEA
4.  % theta --- 0.2 --- Penalty parameter
5.  % fPFE  --- 0.1 --- Frequency of generic front modeling
```

Line 2 gives two labels of the function, the first label `<algorithm>` indicates that this is an MOEA function (it is `<problem>` and `<metric>` for MOP classes and performance metric functions, respectively), and the second label `<G>` can be an arbitrary string. Line 3 gives the full name of the MOEA. Lines 4-5 define the parameters for GFM-MOEA, where the name of the parameter is located on the first

column of each line, the default value is located on the second column, and the introduction is located on the third column; the columns are divided by `---`.

GFM-MOEA then receives the parameter settings from users by the following command:

```
[theta,fPFE] = Global.ParameterSet(0.2,0.1);
```

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

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, and the number of evaluated individuals `GLOBAL.evaluated` will not be increased.

```
OffDec = GA(Population(MatingPool).decs);
```

For the function `GA()`, if the input is an array of `INDIVIDUAL` objects, the output is also an array of `INDIVIDUAL` objects. While if the input is a matrix of decision variables, the output is also a matrix of decision variables.

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

```
classdef DTLZ2 < PROBLEM
    methods
        function obj = DTLZ2()
            if isempty(obj.Global.M)
                obj.Global.M = 3;
            end
            if isempty(obj.Global.D)
                obj.Global.D = obj.Global.M + 9;
            end
            obj.Global.lower = zeros(1,obj.Global.D);
            obj.Global.upper = ones(1,obj.Global.D);
            obj.Global.encoding = 'real';
        end
        function PopObj = CalObj(obj,PopDec)
```

```
            M = obj.Global.M;
            g = sum((PopDec(:,M:end)-0.5).^2,2);
            PopObj = repmat(1+g,1,M).*fliplr(cumprod([ones
            (size(g,1),1),cos(PopDec(:,1:M-1)*pi/2)],2)).*
            [ones(size(g,1),1),sin(PopDec(:,M-1:-1:1)*pi/2)];
        end
        function P = PF(obj,N)
            P = UniformPoint(N,obj.Global.M);
            P = P./repmat(sqrt(sum(P.^2,2)),1,obj.Global.M);
        end
    end
end
```

To begin with, `DTLZ2` is a subclass of `PROBLEM`, and each operation is an overloaded method. In the constructor `DTLZ2.DTLZ2()`, the values of number of objective `GLOBAL.M`, the number of decision variables `GLOBAL.D`, the lower bounds of decision variables `GLOBAL.lower`, the upper bounds of decision variables `GLOBAL.upper`, and the encoding `GLOBAL.encoding` are set. In `DTLZ2.CalObj()`, the objective values of a population are calculated. In `DTLZ2.PF()`, a set of reference points on the true Pareto front is sampled.

It is worth to note that a decision variable may be infeasible in three cases. Firstly, 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, while MOP classes need not handle this case. Secondly, it may not fulfill the constraints (a positive constraint violation indicates that this constraint is not fulfilled), in this case the constraint violations should be calculated by overloading the method `PROBLEM.CalCon()`, for example in `C1_DTLZ1.m`,

```
function PopCon = CalCon(obj,PopDec)
    PopObj = obj.CalObj(PopDec);
    PopCon = PopObj(:,end)/0.6+sum(PopObj(:,1:end-1)/0.5,2)
    -1;
end
```

Thirdly, for combinational MOPs, it may be an illegal character, in this case it should be repaired by overloading the method `PROBLEM.CalDec()`, for example in `MOKP.m`,

```
function PopDec = CalDec(obj,PopDec)
    C = sum(obj.W,2)/2;
    [~,rank] = sort(max(obj.P./obj.W));
    for i = 1 : size(PopDec,1)
        while any(obj.W*PopDec(i,:)'>C)
            k = find(PopDec(i,rank),1);
            PopDec(i,rank(k)) = 0;
        end
    end
end
```

Similar to MOEA functions, MOP classes can receive parameter settings from users, for example in `WFG1.m`, the following command is used to receive parameter settings:

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

D. Add Performance Metrics

A performance metric 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

```
function score = IGD(PopObj,PF)
% <metric> <min>
    score = mean(min(pdist2(PF,PopObj),[],2);
end
```

To begin with, a performance metric function has two input parameters and one output parameter, where `PopObj` denotes the matrix of objective values of a population, `PF` denotes a set of reference points sampled on the true Pareto front, and `score` denotes the performance metric value. Note that the comment in Line 2 is necessary for being identified by the GUI, where the first label `<metric>` indicates that this is a performance metric function, and the second label `<min>` indicates that a smaller metric value means a better performance. By contrast, if the second label is `<max>`, it indicates that a larger metric value means a better performance.