PlatEMO: A MATLAB Platform for Evolutionary Multi-Objective Optimization

Ye Tian, School of Computer Science and Technology, Anhui University, Hefei, China
Ran Cheng, School of Computer Science, University of Birmingham, Birmingham, U.K.
Xingyi Zhang, School of Computer Science and Technology, Anhui University, Hefei, China
Yaochu Jin, Department of Computer Science, University of Surrey, Guildford, U.K.

Abstract

Over the last three decades, a large number of evolutionary algorithms have been developed for solving multi-objective optimization problems. However, there lacks an up-to-date and comprehensive software platform for researchers to properly benchmark existing algorithms and for practitioners to apply selected algorithms to solve their real-world problems. The demand of such a common tool becomes even more urgent, when the source code of many proposed algorithms has not been made publicly available. To address these issues, we have developed a MATLAB platform for evolutionary multi-objective optimization in this paper, called PlatEMO, which includes more than 50 multi-objective evolutionary algorithms and more than 100 multi-objective test problems, along with several widely used performance indicators. With a user-friendly graphical user interface, PlatEMO enables users to easily compare several evolutionary algorithms at one time and collect statistical results in Excel or LaTeX files. More importantly, PlatEMO is completely open source, such that users are able to develop new algorithms on the basis of it. This paper introduces the main features of PlatEMO and illustrates how to use it for performing comparative experiments, embedding new algorithms, creating new test problems, and developing performance indicators. Source code of PlatEMO is now available at: http://bimk.ahu.edu.cn/index.php?s=/Index/Software/index.html.

I. INTRODUCTION

Multi-objective optimization problems (MOPs) widely exist in computer science such as data mining [1], pattern recognition [2], image processing [3] and neural network [4], as well as many other application fields [5]–[8]. An MOP consists of two or more conflicting objectives to be optimized, and there often exist a set of optimal solutions trading off between different objectives. Since the vector evaluated genetic algorithm (VEGA) was proposed by Schaffer in 1985 [9], a number of multi-objective evolutionary algorithms (MOEAs) have been proposed and shown their superiority in tackling MOPs during the last three decades. For example, several MOEAs based on Pareto ranking selection and fitness sharing mechanism including multi-objective genetic algorithm (MOGA) [10], non-dominated sorting genetic algorithm (NSGA) [11], and niched Pareto genetic algorithm (NPGA) [12] were proposed in the 1990s. From 1999 to 2002, some MOEAs characterized by the elitism strategy were developed, such as non-dominated sorting genetic algorithm II (NSGA-II) [13], strength Pareto evolutionary algorithm 2 (SPEA2) [14], Pareto envelope-based selection algorithm II (PESA-II) [15] and cellular multiobjective genetic algorithm (cellular MOGA) [16]. Afterwards, the evolutionary algorithm based

Corresponding Author: Xingyi Zhang (E-mail: xyzhangghust@gmail.com)
on decomposition (MOEA/D) was proposed in 2007 [17], and some other MOEAs following the basic idea of MOEA/D had also been developed since then [18]–[21].

In spite of a large number of MOEAs in the literature [22], there often exist some difficulties in applying and using these algorithms since the source code of most algorithms had not been provided by the authors. Besides, it is also difficult to make benchmark comparisons between MOEAs due to the lack of a general experimental environment. To address such issues, several MOEA libraries have been proposed to provide uniform experimental environments for users [23]–[30], which have greatly advanced the multi-objective optimization research and the implementation of new ideas. For example, the C-based multi-objective optimization library PISA [27] separates the implementation into two components, i.e., the problem-specific part containing MOPs and operators, and the problem-independent part containing MOEAs. These two components are connected by a text file-based interface in PISA. jMetal [23] is an object-oriented Java-based multi-objective optimization library consisting of various MOEAs and MOPs. MOEA Framework is another free and open source Java framework for multi-objective optimization, which provides a comprehensive collection of MOEAs and tools necessary to rapidly design, develop, execute and test MOEAs. OTL [25], a C++ template library for multi-objective optimization, is characterized by object-oriented architecture, template technique, ready-to-use modules, automatically performed batch experiments and parallel computing. Besides, a Python-based experimental platform has also been proposed as the supplement of OTL, for improving the development efficiency and performing batch experiments more conveniently. ParadisEO-MOEO [26] is also a notable MOEA library designed on the basis of reconfigurable components. It provides a wide range of archive-related features and fitness assignment strategies used in the most common Pareto-based evolutionary algorithms, such that users can use the library to generate a large number of new MOEAs by recombining these components. AutoMOEA [30] is another recently proposed MOEA template extended from ParadisEO-MOEO, which has a higher generality and more comprehensive coverage of algorithms and operators. In addition, a similar framework was also adopted in ParadisEO-MO for the design of single solution-based metaheuristics [31].

It is encouraging that there are several MOEA libraries dedicated to the development of evolutionary multi-objective optimization (EMO), but unfortunately, most of them are still far from useful and practical to most researchers. Besides, due to the lack of professional GUI for experimental settings and algorithmic configurations, these libraries are difficult to be used or extended, especially for beginners who are not familiar with EMO. In order to collect more modern MOEAs and make the implementation of experiments on MOEAs easier, in this paper, we introduce a MATLAB-based EMO platform called PlatEMO. Compared to existing EMO platforms, PlatEMO has the following main advantages:

- **Rich Library.** PlatEMO now includes 50 existing popular MOEAs published in important journals or conferences in evolutionary computation community as shown in Table I, which cover a variety of different types, including multi-objective genetic algorithms, multi-objective differential evolution algorithms, multi-objective particle swarm optimization algorithms, multi-objective estimation of distribution algorithms, and surrogate-assisted multi-objective evolutionary algorithms. PlatEMO also contains 110 MOPs from 16 popular test suites covering various difficulties, which are listed in Table II. In addition, there are a lot of performance indicators provided by PlatEMO for experimental studies, includingCoverage [75], generational distance (GD) [89], hypervolume (HV) [90], inverted generational distance (IGD) [91], normalized
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Year of Publication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multi-Objective Genetic Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEA2 [14]</td>
<td>2001</td>
<td>Strength Pareto evolutionary algorithm 2</td>
</tr>
<tr>
<td>ε-MOEA [32]</td>
<td>2003</td>
<td>Multi-objective evolutionary algorithm based on ε-dominance</td>
</tr>
<tr>
<td>IBEA [33]</td>
<td>2004</td>
<td>Indicator-based evolutionary algorithm</td>
</tr>
<tr>
<td>SMS-EMOA [34]</td>
<td>2007</td>
<td>S metric selection evolutionary multi-objective optimization algorithm</td>
</tr>
<tr>
<td>MSOPS-II [35]</td>
<td>2007</td>
<td>Multiple single objective Pareto sampling algorithm II</td>
</tr>
<tr>
<td>MTS [36]</td>
<td>2009</td>
<td>Multiple trajectory search</td>
</tr>
<tr>
<td>AGE-II [37]</td>
<td>2013</td>
<td>Approximation-guided evolutionary algorithm II</td>
</tr>
<tr>
<td>NSLS [38]</td>
<td>2015</td>
<td>Non-dominated sorting and local search</td>
</tr>
<tr>
<td>BCE-IBEA [39]</td>
<td>2015</td>
<td>Bi-criterion evolution for IBEA</td>
</tr>
<tr>
<td>MOEA/IGD-NS [40]</td>
<td>2016</td>
<td>Multi-objective evolutionary algorithm based on an enhanced inverted generational distance metric</td>
</tr>
<tr>
<td><strong>Many-Objective Genetic Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HypE [41]</td>
<td>2011</td>
<td>Hypervolume-based estimation algorithm</td>
</tr>
<tr>
<td>PICEA-g [42]</td>
<td>2013</td>
<td>Preference-inspired coevolutionary algorithm with goals</td>
</tr>
<tr>
<td>GtEA [43]</td>
<td></td>
<td>Grid-based evolutionary algorithm</td>
</tr>
<tr>
<td>SPEA2+SDE [46]</td>
<td>2014</td>
<td>SPEA2 with shift-based density estimation</td>
</tr>
<tr>
<td>BiGE [47]</td>
<td>2015</td>
<td>Bi-goal evolution</td>
</tr>
<tr>
<td>EFR-RR [20]</td>
<td>2015</td>
<td>Ensemble fitness ranking with ranking restriction</td>
</tr>
<tr>
<td>I-DBEA [48]</td>
<td>2015</td>
<td>Improved decomposition based evolutionary algorithm</td>
</tr>
<tr>
<td>KtEA [49]</td>
<td>2015</td>
<td>Knee point driven evolutionary algorithm</td>
</tr>
<tr>
<td>MaOEA-DDFC [50]</td>
<td>2015</td>
<td>Many-objective evolutionary algorithm based on directional diversity and favorable convergence</td>
</tr>
<tr>
<td>MOEA/ADD [51]</td>
<td>2015</td>
<td>Multi-objective evolutionary algorithm based on dominance and decomposition</td>
</tr>
<tr>
<td>MOSMI-II [52]</td>
<td>2015</td>
<td>Many-objective metaheuristic based on the R2 indicator II</td>
</tr>
<tr>
<td>Two_Arch2 [53]</td>
<td>2015</td>
<td>Two-archive algorithm 2</td>
</tr>
<tr>
<td>MaOEA-R&amp;D [54]</td>
<td>2016</td>
<td>Many-objective evolutionary algorithm based on objective space reduction and diversity improvement</td>
</tr>
<tr>
<td>RPEA [55]</td>
<td>2016</td>
<td>Reference points-based evolutionary algorithm</td>
</tr>
<tr>
<td>RVEA [56]</td>
<td>2016</td>
<td>Reference vector guided evolutionary algorithm</td>
</tr>
<tr>
<td>RVEA* [56]</td>
<td>2016</td>
<td>RVEA embedded with the reference vector regeneration strategy</td>
</tr>
<tr>
<td>SPEAR [57]</td>
<td>2016</td>
<td>Strength Pareto evolutionary algorithm based on reference direction</td>
</tr>
<tr>
<td>θ-DEA [58]</td>
<td>2016</td>
<td>θ-dominance based evolutionary algorithm</td>
</tr>
<tr>
<td><strong>Multi-objective Genetic Algorithms for Large-Scale Optimization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOEA/DVA [59]</td>
<td>2016</td>
<td>Multi-objective evolutionary algorithm based on decision variable analyses</td>
</tr>
<tr>
<td>LMEA [60]</td>
<td>2016</td>
<td>Large-scale many-objective evolutionary algorithm</td>
</tr>
<tr>
<td><strong>Multi-Objective Genetic Algorithms with Preference</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g-NSGA-II [61]</td>
<td>2009</td>
<td>g-dominance based NSGA-II</td>
</tr>
<tr>
<td>r-NSGA-II [62]</td>
<td>2010</td>
<td>r-dominance based NSGA-II</td>
</tr>
<tr>
<td><strong>Multi-objective Differential Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDE3 [64]</td>
<td>2005</td>
<td>Generalized differential evolution 3</td>
</tr>
<tr>
<td><strong>Multi-Objective Particle Swarm Optimization Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOPSO [65]</td>
<td>2002</td>
<td>Multi-objective particle swarm optimization</td>
</tr>
<tr>
<td>SMPSO [66]</td>
<td>2005</td>
<td>Speed-constrained multi-objective particle swarm optimization</td>
</tr>
<tr>
<td>dMOPSO [67]</td>
<td>2011</td>
<td>Decomposition-based particle swarm optimization</td>
</tr>
<tr>
<td><strong>Multi-Objective Memetic Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M-PAES [68]</td>
<td>2000</td>
<td>Memetic algorithm based on Pareto archived evolution strategy</td>
</tr>
<tr>
<td><strong>Multi-Objective Estimation of Distribution Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MO-CMA [69]</td>
<td>2007</td>
<td>Multi-objective covariance matrix adaptation</td>
</tr>
<tr>
<td>RM-MEDA [70]</td>
<td>2008</td>
<td>Regularity model-based multi-objective estimation of distribution algorithm</td>
</tr>
<tr>
<td>IM-MOEAD [71]</td>
<td>2015</td>
<td>Inverse modeling multi-objective evolutionary algorithm</td>
</tr>
<tr>
<td><strong>Surrogate-Assisted Multi-Objective Evolutionary Algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParEGO [72]</td>
<td>2005</td>
<td>Efficient global optimization for Pareto optimization</td>
</tr>
<tr>
<td>SMS-EGO [73]</td>
<td>2008</td>
<td>S-metric-selection-based efficient global optimization</td>
</tr>
<tr>
<td>K-RVEA [74]</td>
<td>2016</td>
<td>Kriging assisted RVEA</td>
</tr>
</tbody>
</table>
TABLE II
THE 110 MULTI-OBJECTIVE OPTIMIZATION PROBLEMS INCLUDED IN THE CURRENT VERSION OF PlatEMO.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Year of Publication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOKP [75]</td>
<td>1999</td>
<td>Multi-objective 0/1 knapsack problem and behavior of MOEAs on this problem analyzed in [76]</td>
</tr>
<tr>
<td>ZDT1–ZDT6 [77]</td>
<td>2000</td>
<td>Multi-objective test problems</td>
</tr>
<tr>
<td>mQAP [78]</td>
<td>2003</td>
<td>Multi-objective quadratic assignment problem</td>
</tr>
<tr>
<td>DTLZ1–DTLZ9 [79]</td>
<td>2005</td>
<td>Scalable multi-objective test problems</td>
</tr>
<tr>
<td>WFG1–WFG9 [80]</td>
<td>2006</td>
<td>Scalable multi-objective test problems and degenerate problem WFG3 analyzed in [81]</td>
</tr>
<tr>
<td>MONRP [82]</td>
<td>2007</td>
<td>Multi-objective next release problem</td>
</tr>
<tr>
<td>MOTSP [83]</td>
<td>2007</td>
<td>Multi-objective traveling salesperson problem</td>
</tr>
<tr>
<td>Pareto-Box [84]</td>
<td>2007</td>
<td>Pareto-Box problem</td>
</tr>
<tr>
<td>CF1–CF10 [85]</td>
<td>2008</td>
<td>Constrained multi-objective test problems for the CEC 2009 special session and competition</td>
</tr>
<tr>
<td>F1–F10 for RM-MEDA [70]</td>
<td>2008</td>
<td>The test problems designed for RM-MEDA</td>
</tr>
<tr>
<td>UF1–UF12 [85]</td>
<td>2008</td>
<td>Unconstrained multi-objective test problems for the CEC 2009 special session and competition</td>
</tr>
<tr>
<td>C1_DTLZ1, C2_DTLZ2, C3_DTLZ4, DTLZ1, DTLZ2 [45]</td>
<td>2014</td>
<td>Constrained DTLZ and inverted DTLZ</td>
</tr>
<tr>
<td>F1–F10 for IM-MOEA [71]</td>
<td>2015</td>
<td>The test problems designed for IM-MOEA</td>
</tr>
<tr>
<td>BT1–BT9 [87]</td>
<td>2016</td>
<td>Multi-objective test problems with bias</td>
</tr>
<tr>
<td>LSMOP1–LSMOP9 [88]</td>
<td>2016</td>
<td>Large-scale multi-objective test problems</td>
</tr>
</tbody>
</table>

hypervolume (NHV) [41], pure diversity (PD) [92], spacing [93], spread (Δ) [94], and the performance metric ensemble [95]. PlatEMO also provides a lot of widely-used operators for different encodings [96]–[102], which can be used together with all the MOEAs in PlatEMO.

- **Good Usability.** PlatEMO is fully developed in MATLAB language, thus any machines installed with MATLAB can use PlatEMO regardless of the operating system. Besides, users do not need to write any additional code when performing experiments using MOEAs in PlatEMO, as PlatEMO provides a user-friendly GUI, where users can configure all the settings and perform experiments on MOEAs via the GUI, and the experimental results can further be saved as a table in the format of Excel or LaTeX. In other words, with the assistance of PlatEMO, users can directly obtain the statistical experimental results to be used in academic writings by one-click operation via the GUI.

- **Easy Extensibility.** PlatEMO is not only easy to be used, but also easy to be extended. To be specific, the source code of all the MOEAs, MOPs and operators in PlatEMO are completely open source, and the length of the source code is very short due to the advantages of matrix operation in MATLAB, such that users can easily implement their own MOEAs, MOPs and operators on the basis of existing resources in PlatEMO. In addition, all new MOEAs developed on the basis of interfaces provided by PlatEMO can be also included in the platform, such that the library in PlatEMO can be iteratively updated by all users to follow state-of-the-arts.

- **Delicate Considerations.** There are many delicate considerations in the implementation of PlatEMO. For example, PlatEMO provides different figure demonstrations of experimental results, and it also provides well-designed sampling methods which are able to generate an arbitrary number of uniformly distributed
Fig. 1. The reference points generated by PlatEMO on the Pareto fronts of CF8–10, DTLZ1–9, UF8–10 and WFG1–3 with 3 objectives. Note that the reference points do not cover the whole true Pareto front of WFG3, since the geometry of the true Pareto front is still unknown [81].

points on the Pareto optimal fronts with an arbitrary number of objectives. Fig. 1 shows the reference points sampled by PlatEMO on the Pareto optimal fronts of some MOPs with 3 objectives, while such reference points have not been provided by any other existing EMO libraries. It is also worth noting that, since the efficiency of most MOEAs is subject to the non-dominated sorting process, PlatEMO employs the efficient non-dominated sort ENS-SS [103] for two- and three-objective optimization and the tree-based ENS termed T-ENS [60] for optimization with more than three objectives as the non-dominated sorting approaches, which have been demonstrated to be much more efficient than the widely adopted fast non-dominated sorting approach [13] as well as other popular non-dominated sorting approaches [104].

Table III lists the main characteristics of PlatEMO in comparison with other existing MOEA libraries. As shown in the table, PlatEMO provides a variety of MOEAs and MOPs with different types, where the types of MOEAs cover genetic algorithm, differential algorithm, particle swarm optimization, memetic algorithm, estimation of distribution algorithm, surrogate-assisted evolutionary algorithm and the types of MOPs have multi-objective, many-objective, combinatorial, large-scale and expensive optimization problem. The PlatEMO shows significant superiority in usability of MOEAs and extendibility of adding new MOEAs, while ParadisEO-MOEO has the best configurability in terms of generating new MOEAs by combining different components.

The rest of this paper is organized as follows. In the next section, the architecture of PlatEMO is presented on several aspects, i.e., the file structure of PlatEMO, the class diagram of PlatEMO, and the sequence diagram of executing algorithms by PlatEMO. Section III introduces how to use PlatEMO for analyzing the performance of algorithms and performing comparative experiments. The methods of extending PlatEMO with new MOEAs,
TABLE III
COMPARISON BETWEEN PLATEMO AND FIVE EXISTING MOEA LIBRARIES.

<table>
<thead>
<tr>
<th>MOEA Library</th>
<th>Language</th>
<th>Types of MOEAs Available</th>
<th>Types of MOPs Available</th>
<th>Usability</th>
<th>Components Configurability</th>
<th>Extendibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParadisEO-MOE [26]</td>
<td>C++</td>
<td>Genetic algorithm, Simulated annealing, Tabu search</td>
<td>Multi-objective, Combinatorial</td>
<td>Normal</td>
<td>High</td>
<td>Normal</td>
</tr>
<tr>
<td>PISA [27]</td>
<td>C</td>
<td>Genetic algorithm</td>
<td>Multi-objective, Many-objective, Combinatorial</td>
<td>Normal</td>
<td>Low</td>
<td>Normal</td>
</tr>
<tr>
<td>jMetal [23]</td>
<td>Java</td>
<td>Genetic algorithm, Differential algorithm, Particle swarm optimization</td>
<td>Multi-objective, Many-objective, Combinatorial</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>MOEA Framework</td>
<td>Java</td>
<td>Genetic algorithm, Differential algorithm, Particle swarm optimization</td>
<td>Multi-objective, Many-objective, Combinatorial</td>
<td>Normal</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>PlatEMO</td>
<td>MATLAB</td>
<td>Genetic algorithm, Differential algorithm, Particle swarm optimization, Memetic algorithm, Estimation of distribution algorithm, Surrogate-assisted evolutionary algorithm</td>
<td>Multi-objective, Many-objective, Combinatorial, Large-scale, Expensive</td>
<td>High</td>
<td>Normal</td>
<td>High</td>
</tr>
</tbody>
</table>


MOPs, operators and performance indicators are described in Section IV. Finally, conclusion and future work are given in Section V.

II. ARCHITECTURE OF PLATEMO

After opening the root directory of PlatEMO, users can see a lot of .m files organized in the structure shown in Fig. 2, where it is very easy to find the source code of specified MOEAs, MOPs, operators or performance indicators. As shown in Fig. 2, there are six folders and one interface function main.m in the root directory of PlatEMO. The first folder \Algorithms is used to store all the MOEAs in PlatEMO, where each MOEA has an independent subfolder and all the relevant functions are in it. For instance, as shown in Fig. 2, the subfolder \Algorithms\NSGA-II contains three functions NSGAIIm, CrowdingDistance.m and EnvironmentalSelection.m, which are used to perform the main loop, calculate the crowding distances, and perform the environmental selection of NSGA-II, respectively. The second folder \Problems contains a lot of subfolders for storing benchmark MOPs. For example, the subfolder \Problems\DTLZ contains 15 DTLZ test problems (i.e., DTLZ1–DTLZ9, C1_DTLZ1, C2_DTLZ2, C3_DTLZ4, IDTLZ1, IDTLZ2 and CDTLZ2), and the subfolder \Problems\WFG contains 9 WFG test problems (i.e., WFG1–WFG9). The folders \Operators and \Metrics store all the operators and performance indicators, respectively. The next folder \Public is used to store some public classes and functions, such as GLOBAL.m and INDIVIDUAL.m, which are two classes in
PlatEMO representing settings of parameters and definitions of individuals, respectively. The last folder \GUI stores all the functions for establishing the GUI of PlatEMO, where users need not read or modify them.

PlatEMO also has a simple architecture, where it only involves two classes, namely GLOBAL and INDIVIDUAL, to store all the parameters and joint all the components (e.g., MOEAs, MOPs and operators). The class diagram of these two classes is presented in Fig. 3. The first class GLOBAL represents all the parameter setting,
including the handle of MOEA function algorithm, the handle of MOP function problem, the handle of operator function operator and other parameters about the environment (the population size, the number of objectives, the length of decision variables, the maximum number of fitness evaluations, etc.). Note that all the properties in GLOBAL are read-only, which can only be assigned by users when the object is being instantiated. GLOBAL also provides several methods to be invoked by MOEAs, where MOEAs can achieve some complex operations via these methods. For instance, the method Initialization() can generate a randomly initial population with specified size, and the method Variation() can generate a set of offsprings with specified parents.

The other class in PlatEMO is INDIVIDUAL, where its objects are exactly individuals in MOEAs. An INDIVIDUAL object contains four properties, i.e., dec, obj, con and add, all of which are also read-only. dec is the array of decision variables of the individual, which is assigned when the object is being instantiated. obj and con store the objective values and the constraint values of the individual, respectively, which are calculated after dec has been assigned. The property add is used to store additional properties of the individual for some special operators, such as the ‘speed’ property in PSO operator [101].

In order to better understand the mechanism of PlatEMO, Fig. 4 illustrates the sequence diagram of running an MOEA by PlatEMO without GUI. To begin with, the interface main.m first invokes the algorithm function (e.g., NSGAIL.m), then the algorithm obtains an initial population (i.e., an array of INDIVIDUAL objects) from the GLOBAL object by invoking its method Initialization(). After that, the algorithm starts the evolution until the
termination criterion is fulfilled, where the maximum number of fitness evaluations is used as the termination
criterion for all the MOEAs in PlatEMO. In each generation of a general MOEA, it first performs mating pool
selection for selecting a number of parents from the current population, and the parents are used to generate
offsprings by invoking the method Variation() of GLOBAL object. Variation() then passes the parents to the
operator function (e.g., DE.m), which is used to calculate the decision variables of the offsprings according
to the parents. Next, the operator function invokes the INDIVIDUAL class to instantiate the offspring objects,
where the objective values of offsprings are calculated by invoking the problem function (e.g., DTLZ1.m). After
obtaining the offsprings, the algorithm performs environmental selection on the current population and the
offsprings to select the population for next generation. When the number of instantiated INDIVIDUAL objects
exceeds the maximum number of fitness evaluations, the algorithm will be terminated and the final population
will be output.

As presented by the above procedure, the algorithm function, the problem function and the operator function
do not invoke each other directly; instead, they are connected to each other by the GLOBAL class and the
INDIVIDUAL class. This mechanism has two advantages. First, MOEAs, MOPs and operators in PlatEMO
are independent mutually, and they can be arbitrarily combined with each other, thus improved the flexibility
of PlatEMO. Second, users need not consider the details of the MOP or the operator to be involved when
developing a new MOEA, thus significantly improving the development efficiency.

### III. Running PlatEMO

As mentioned in Section I, PlatEMO provides two ways to run it: first, it can be run with a GUI by invoking
the interface main() without input parameter, then users can perform MOEAs on MOPs by simple one-click
operations; second, it can be run without GUI, and users can perform one MOEA on an MOP by invoking
main() with input parameters. In this section, we elaborate these two ways of running PlatEMO.

#### A. Running PlatEMO without GUI

The interface main() can be invoked with a set of input parameters by the following form: main('name1',
value1, 'name2', value2, ...), where name1, name2, ... denote the names of the parameters and value1, value2,
... denote the values of the parameters. All the acceptable parameters together with their data types and default values are listed in Table IV. It is noteworthy that every parameter has a default value so that users need not assign all the parameters. As an example, the command `main(‘-algorithm’,@NSGAII,’-problem’,@DTLZ2,’-N’,100,’-M’,3,’-D’,10,’-evaluation’,10000)` is used to perform NSGA-II on DTLZ2 with a population size of 100, an objective number of 3, a decision variable length of 10, and a maximum fitness evaluation number of 10000.

By invoking `main()` with parameters, one MOEA can be performed on an MOP with the specified setting, while the GUI will not be displayed. After the MOEA has been terminated, the final population will be displayed or saved, which is determined by the parameter `-mode` shown in Table IV. To be specific, if `-mode` is set to 1, the objective values or decision variable values of the final population will be displayed in a new figure, and users can also observe the true Pareto front and the evolutionary trajectories of performance indicator values. For example, Fig. 5 shows the objective values of the population obtained by NSGA-II on DTLZ2 with 3 objectives, where users can select the figure to be displayed on the rightmost menu. If `-mode` is set to 2, the final population will be saved in a `.mat` file, while no figure will be displayed. If `-mode` is set to 3, users can specify any operation to be performed on the final population, e.g., displaying and saving the population concurrently.

Generally, as listed in Table IV, four parameters related to the optimization can be assigned by users (i.e., the population size `-N`, the number of objectives `-M`, the number of decision variables `-D`, and the maximum number of fitness evaluations `-evaluation`); however, different MOEAs, MOPs or operators may involve additional parameter settings. For instance, there is a parameter `rate` denoting the ratio of selected knee points in KnEA [49], and there are four parameters `proC`, `disC`, `proM` and `disM` in EAreal [96], [97], which denote the crossover probability, the distribution index of simulated binary crossover, the number of bits undergone mutation, and the distribution index of polynomial mutation, respectively. In PlatEMO, such function...
related parameters can also be assigned by users via assigning the parameter \( -X_{\text{parameter}} \), where \( X \) indicates the name of the function. For example, users can use the command `main(. . . , '-KnEA_{parameter}',{0.5},. . . )` to set the value of `rate` to 0.5 for `KnEA`, and use the command `main(. . . , '-EAreal_{parameter}',{1,20,1,20},. . . )` to set the values of `proC`, `disC`, `proM` and `disM` to 1, 20, 1 and 20 for `EAreal`, respectively. Besides, users can find the acceptable parameters of each MOEA, MOP and operator in the comments at the beginning of the corresponding function.

B. Running PlatEMO with GUI

The GUI of PlatEMO currently contains two modules. The first module is used to analyze the performance of each MOEA, where one MOEA on an MOP can be performed in this module each time, and users can observe the result via different figure demonstrations. The second module is designed for statistical experiments, where multiple MOEAs on a batch of MOPs can be performed at the same time, and the statistical experimental results can be saved as Excel table or LaTeX table.

The interface of the first module, i.e., test module, is shown in Fig. 6. As can be seen from the figure, the main panel is divided into four parts. The first subpanel from left provides three pop up menus, where users can select the MOEA, MOP and operator to be performed. The second subpanel lists all the parameters to be assigned, which depends on the selected MOEA, MOP and operator. The third subpanel displays the current population during the optimization, other figures such as the true Pareto front and the evolutionary trajectories of performance indicator values can also be displayed. In addition, users can observe the populations in previous generations by dragging the slider at the bottom. The fourth subpanel stores the detailed information of historical
The other module on the GUI is the experimental module, which is shown in Fig. 7. Similar to the test module, users should first select the MOEAs, MOPs and operators to be performed in the leftmost subpanel. Note that multiple MOEAs and MOPs can be selected in the experimental module. After setting the number of total runs, folder for saving results, and all the relevant parameters, the experiment can be started and the statistical results will be shown in the rightmost subpanel. Users can select any performance indicator to calculate the results to

![PlatEMO GUI](image)

**Fig. 7.** The experimental module of PlatEMO.

**TABLE V**

IGD VALUES OF NSGA-III, KNEA AND RVEA ON DTLZ1–DTLZ4. THE LaTeX CODE OF THIS TABLE IS AUTOMATICALLY GENERATED BY PlatEMO.

<table>
<thead>
<tr>
<th>Problem</th>
<th>$M$</th>
<th>NSGA-III</th>
<th>KNEA</th>
<th>RVEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLZ1</td>
<td>2</td>
<td>$1.1556e-1$ (1.49e-1) ≈ $7.5785e-2$ (1.47e-1)</td>
<td>$4.6018e-1$ (5.52e-1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>$2.9794e-1$ (2.85e-1) ≈ $1.8086e-1$ (1.37e-1)</td>
<td>$4.2677e-1$ (2.33e-1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>$4.6226e-1$ (2.66e-1) ≈ $2.6713e-1$ (2.55e-1)</td>
<td>$7.3718e-1$ (7.76e-1)</td>
<td></td>
</tr>
<tr>
<td>DTLZ2</td>
<td>5</td>
<td>$2.1746e-1$ (9.86e-4) ≈ $2.4033e-1$ (1.14e-2)</td>
<td>$2.1394e-1$ (3.65e-4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>$2.8677e-1$ (2.01e-3) ≈ $3.1285e-1$ (9.7e-3)</td>
<td>$2.3203e-1$ (9.06e-4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>$3.5352e-1$ (3.93e-3) ≈ $3.6826e-1$ (7.51e-3)</td>
<td>$3.4742e-1$ (9.55e-4)</td>
<td></td>
</tr>
<tr>
<td>DTLZ3</td>
<td>8</td>
<td>$5.8255e+1$ (3.97e+1) ≈ $1.3282e+2$ (6.02e+1)</td>
<td>$9.2466e+0$ (4.45e+0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>$5.4964e+1$ (1.19e+1) ≈ $2.0923e+2$ (5.21e+1)</td>
<td>$1.3395e+1$ (5.95e+0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>$6.1538e+1$ (4.95e+0) ≈ $2.6730e+2$ (1.10e+2)</td>
<td>$1.2604e+1$ (9.14e+0)</td>
<td></td>
</tr>
<tr>
<td>DTLZ4</td>
<td>11</td>
<td>$6.1257e-1$ (1.03e-1) ≈ $5.3976e-1$ (7.19e-3)</td>
<td>$5.8265e-1$ (3.49e-2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>$7.0350e-1$ (8.27e-2) ≈ $6.1240e-1$ (5.45e-3)</td>
<td>$6.3605e-1$ (2.85e-2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>$6.6993e-1$ (5.79e-2) ≈ $6.0250e-1$ (2.03e-3)</td>
<td>$6.3893e-1$ (2.54e-2)</td>
<td></td>
</tr>
</tbody>
</table>

As a result, the test module provides similar functions to the PlatEMO without GUI, but users do not need to write any additional command or code when using it.

results. As a result, the test module provides similar functions to the PlatEMO without GUI, but users do not need to write any additional command or code when using it.
Fig. 8. The source code of the main function of NSGA-II. The common code required by any MOEA is underlined.

be listed in the table, where the mean and the standard deviation of the performance indicator value are shown in each grid. Furthermore, the best result in each row is shown in blue, and the Wilcoxon rank sum test result is labeled by the signs ‘+’, ‘−’ and ‘≈’, which indicate that the result is significantly better, significantly worst and statistically similar to the result in the control column, respectively. After the experiment is finished, the data shown in the table can be saved as Excel table (.xlsx file) or LaTeX table (.tex file). For example, after obtaining the experimental results shown in the table in Fig. 7, users can press the ‘saving’ button on the GUI to save the table in the format of LaTeX, where the corresponding LaTeX table is shown in Table V.

It can be concluded from the above introduction that the functions provided by PlatEMO are modularized, where two modules (i.e., the test module and the experimental module) are included in the current version of PlatEMO. In the future, we also plan to develop more modules to provide more functions for users.

IV. EXTENDING PLATEMO

PlatEMO is an open platform for scientific research and applications of EMO, hence it allows users to add their own MOEAs, MOPs, operators and performance indicators to it, where users should save the new MOEA, MOP, operator or performance indicator to be added as a MATLAB function (i.e., a .m file) with the specified interface and form, and put it in the corresponding folder. In the following, the methods of extending PlatEMO with a new MOEA, MOP, operator and performance indicator are illustrated by several cases, respectively.

A. Adding New Algorithms to PlatEMO

In order to add a new MOEA to PlatEMO, users only need to slightly modify the input and output of their MATLAB function of the algorithm as required by PlatEMO, and put the function (.m file) in the folder \( \text{xAlgorithms} \) of the root directory. For example, as shown in the file structure in Fig. 2, the three .m files for NSGA-II (i.e., NSGAII.m, CrowdingDistance.m and EnvironmentalSelection.m) are all in the subfolder \( \text{xAlgorithms}\text{/NSGA-II} \). A case study including the source code of the main function of NSGA-II (i.e. NSGAII.m) is given in Fig. 8, where the logic of the function is completely the same to the one shown in Fig. 4.

To begin with, the main function of an MOEA has one input parameter and zero output parameter, where the only input parameter denotes the GLOBAL object for the current run. Then an initial population Population is generated by invoking Global.Initialization(), and the non-dominated front number and the crowding distance of each individual are calculated (line 2–4). In each generation, the function Global.NotTermination() is invoked
to check whether the termination criterion is fulfilled, and the variable Population is passed to this 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).

The common code required by any MOEA is underlined in Fig. 8. In addition to the interface of the function, one MOEA needs to perform at least the following three operations: obtaining an initial population via Global.Initialization(), checking the optimization state and outputting the final population via Global.NotTermination(), and generating offsprings via Global.Variation(), where all these three functions are provided by the GLOBAL object. Apart from the above three common operations, different MOEAs may have different logics and different functions to be invoked.

B. Adding New Problems to PlatEMO

All the .m files of MOP functions are stored in the folder Problems, and one .m file usually indicates one MOP. Fig. 9 gives the source code of DTLZ2, where the common code required by any MOP is underlined. It can be seen from the source code that, the interface of DTLZ2 is more complex than the one of NSGA-II, where the function DTLZ2() includes three input parameters and one output parameter. The input parameter Operation determines the operation to be performed; the parameter Global denotes the GLOBAL object; and the parameter input has different meanings when Operation is set to different values, so does the output parameter.

```
1. function varargout = DTLZ2(Operation,Global,input)
2.     switch Operation
3.     case 'init'
4.         Global.M = 3;
5.         Global.D = Global.M + 9;
6.         Global.lower = zeros(1,Global.D);
7.         Global.upper = ones(1,Global.D);
8.         Global.operator = @Eareal;
9.         PopDec = rand(input,Global.D);
10.    varargout = {PopDec};
11.    case 'value'
12.        PopDec = input;
14.        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 = UniformWeight(input,Global.M);
26.        f = f./repmat(sqrt(sum(f.^2,2)),1,Global.M);
27.    varargout = {f};
28.    end
29.    end
```

Fig. 9. The source code of DTLZ2. The common code required by any MOP is underlined.
Different from the MOEA functions which are invoked only once in each run, an MOP function may be invoked many times for different operations. As shown in Fig. 9, an MOP function contains three independent operations: generating random decision variables (line 3–10), calculating objective values and constraint values (line 11–23), and sampling reference points on the true Pareto front (line 24–27). To be specific, if Operation is set to ‘init’, the MOP function will return the decision variables of a random population with size \( \text{input} \) (line 9–10). Meanwhile, it sets Global.M, Global.D, Global.lower, Global.upper and Global.operator to their default values, which denote the number of objectives, number of decision variables, lower boundary of each decision variable, upper boundary of each decision variable, and the operator function, respectively (line 4–8). When Operation is set to ‘value’, the parameter input will denote the decision variables of a population, and the objective values and constraint values of the population will be calculated and returned according to the decision variables (line 14–23). And if Operation is set to ‘PF’, a number of input uniformly distributed reference points will be sampled on the true Pareto front and returned (line 25–27).

**C. Adding New Operators or Performance Indicators to PlatEMO**

Fig. 10 shows the source code of evolutionary operator based on binary coding (i.e. \text{EA}binary.m), where the \text{.m} files of the operator functions are all stored in the folder \text{\textbackslash Operators}. An operator function has two input parameters, one denoting the GLOBAL object (i.e. Global) and the other denoting the parent population (i.e. Parent), and it also has one output parameter denoting the generated offsprings (i.e. Offspring). As can be seen from the source code in Fig. 10, the main task of an operator function is to generate offsprings according to the values of Parent, where \text{EA}binary() performs the single-point crossover in line 6–11 and the bitwise mutation in line 12–13 of the code. Afterwards, the INDIVIDUAL objects of the offsprings are generated and returned (line 14).

Fig. 11 shows the source code of IGD, where all these performance indicator functions are stored in the folder \text{\textbackslash Metrics}. The task of a performance indicator is to calculate the indicator value of a population according to a set of reference points. The input parameters of IGD() consists of two parts: the objective values of the

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

Fig. 10. The source code of evolutionary operator based on binary coding. The common code required by any operator is underlined.
Fig. 11. The source code of IGD. The common code required by any performance indicator is underlined.

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

Fig. 12. The comments and the source code in the head of the function of evolutionary operator based on real value coding.

population (i.e. PopObj), and the reference points sampled on the true Pareto front (i.e. PF). Correspondingly, the output parameter of IGD() is the IGD value (i.e. score). Thanks to the merits of matrix operation in MATLAB, the source code of IGD is quite short as shown in Fig. 11, where the calculation of the mean value of the minimal distance of each point in PF to the points in PopObj can be performed using a built-in function pdist2() provided by MATLAB.

D. Adding Acceptable Parameters for New Functions

All the user-defined functions can have their own parameters as well as the functions provided by PlatEMO, where these parameters can be either assigned by invoking `main(...,'X_parameter','...')` with X denoting the function name, or displayed on the GUI for assignment. In order to add acceptable parameters for an MOEA, MOP, operator or performance indicator function, the comments in the head of the function should be written in a specified form. To be specific, Fig. 12 shows the comments and the source code in the head of the function of evolutionary operator based on real value coding (i.e. EAreal.m).

The comment in line 2 of Fig. 12 gives the two labels of this function, which are used to make sure this function can be identified by the GUI. The comment in line 3 is a brief introduction about this function; for an MOEA or MOP function, such introduction should be the title of the relevant literature. The parameters proC, disC, proM and disM for this function are given by the comments in line 4–7, where the names of the parameters are in the first column, the default values of the parameters are in the second column, and the introductions about the parameters are given in the third column. The columns in each row are divided by the sign ‘—’.

The comments define the parameters and their default values for the function, and invoking `Global.ParameterSet()` can make these parameters assignable to users. As shown in line 9 of Fig. 12, the function invokes `Global.ParameterSet()` with four inputs denoting the default values of the parameters, and sets the four parameters to the outputs. More specifically, if users have not assigned the parameters, they will equal to their default values (i.e. 1, 15, 1 and 15). Otherwise, if users assign the parameters by invoking `main(...,'EAreal_parameter','...')`, the
parameters $proC$, $disC$, $proM$ and $disM$ will be set to $a$, $b$, $c$ and $d$, respectively.

V. CONCLUSION AND FUTURE WORK

This paper has introduced a MATLAB-based open source platform for evolutionary multi-objective optimization, namely PlatEMO. The current version of PlatEMO includes 50 multi-objective optimization algorithms and 110 multi-objective test problems, having covered the majority of state-of-the-arts. Since PlatEMO is developed on the basis of a light architecture with simple relations between objects, it is very easy to be used and extended. Moreover, PlatEMO provides a user-friendly GUI with a powerful experimental module, where engineers and researchers can use it to quickly perform their experiments without writing any additional code. This paper has described the architecture of PlatEMO, and it has also introduced the steps of running PlatEMO with and without the GUI. Then, the ways of adding new algorithms, problems, operators and performance indicators to PlatEMO have been elaborated by several cases.

It is worth noting that, despite that we have performed careful validation and test on the source code of PlatEMO, there may still exist incorrect re-implementations or bugs. To address this issue, we have carefully followed the advice in dealing with algorithm implementations in [105]. We have performed unit tests on every algorithm in PlatEMO thoroughly, highlighted the source of the re-using code, and reported the version number and the change log on the website after each update as suggested in [105]. In addition, since all of the algorithms in PlatEMO are completely open-source, we sincerely hope that users of the software could also join us to make further improvements.

We will continuously maintain and develop PlatEMO in the future. On one hand, we will keep following the state-of-the-arts and adding more effective algorithms and new problems into PlatEMO. On the other hand, more modules will be developed to provide more functions for users, such as preference optimization, dynamic optimization, noisy optimization, etc. We hope that PlatEMO is helpful to the researchers working on evolutionary multi-objective optimization, and we sincerely encourage peers to join us to shape the platform for better functionality and usability.

ACKNOWLEDGEMENT

This work was supported in part by National Natural Science Foundation of China (Grant No. 61672033, 61272152, 615012004, 61502001), and the Joint Research Fund for Overseas Chinese, Hong Kong and Macao Scholars of the National Natural Science Foundation of China (Grant No. 61428302). This manuscript was written during Y. Tian’s visit at the Department of Computer Science, University of Surrey. The authors would like to thank Mr. Kefei Zhou and Mr. Ran Xu for their valued work in testing the PlatEMO.

REFERENCES


