

# A guideline for using the code EPDE

Xiaoyu He\*, Yuren Zhou, and Zefeng Chen

December 8, 2017

## 1 Introduction

This java project contains the detailed implementations of MOEA/D-EP and NSGAIII-EP proposed in the paper “An Evolution Path Based Reproduction Operator for Many-Objective Optimization”, which is in a third submission to TEVC journal.

### 1.1 What we can do with this code?

Users can reproduce some of the experimental data mentioned in the paper.

- Generate an excel file containing all results which are consistent with the experimental data summarized in the paper.
  - Provide the median and IQR results measured by HV, IGD, and  $\Delta_p$  metrics
  - The best and the second best results are highlighted in dark gray and light gray background, respectively
  - The Wilcoxon test is conducted to test whether there is a significant difference between the target algorithm and the peer competitor
  - Provide the averaged ranking results based on the Friedman test
- Run MOEA/D-EP or NSGAIII-EP on a selected problem or a test suite
  - Test the proposed algorithms in a single run
  - Immediately output the IGD value
- Calculate the IGD value for a given algorithm
- Run an experiment automatically
  - MOEA/D-EP and NSGAIII-EP are available
  - Test problems contain DTLZ1-DTLZ7, WFG1-WFG9, and the minus version of DTLZ1-DTLZ4

---

\*email:hxyokokok@foxmail.com

## 1.2 What we can not do?

Things contained in the paper but can not be reproduced with this code:

- Run the competitor algorithms including MOEA/D-SBX, MOEA/D-DE, MOEA/D-BLX, MOEA/D-HOP, NSGAIII, RVEA, MOEA/DD, MOMBI2, MOEA/D-DU, and VaEA. Their codes are not contained in this project. However, their corresponding experimental data, i.e., IGD, HV, and  $\Delta_p$  values, are contained and can be used.
- Calculate the HV and  $\Delta_p$  metrics. The related codes are written in Matlab. Interested readers may contact with us and we will send them through email.

## 2 Requirement

### 2.1 Java runtime

Java 8 is required since the lambda expression is extensively used in the project.

### 2.2 jMetal

Our experiment is implemented on the jMetal 5.3 framework. Using other version of jMetal is not recommended.

### 2.3 Apache POI

Apache POI is required in exporting the experimental data to excel files. The 3.15 version is used in this project.

## 3 Project Structure and Installation

This project is implemented with IntelliJ IDEA community edition. So it is recommended that users also use this IDE to run the project.

### 3.1 Project structure

There are three important directories in the project.

- `libs`. Contain the required libraries of jMetal and Apache POI.
- `out`. Contain all compiled java bytecode files. When running the experiment, the data will be outputted to a subdirectory named “results”. For example, “\$projectBaseDirection\$/EPDE/out/production/EPDE/results”.
- `src`. Contain all source files.
  - `hxy`. A java package contains all “.java” files.

- pf. Contain all reference points well-distributed over the PF for all problems.
- weights.. Contain the reference vectors used in decomposition based algorithms.

## 3.2 Installation

Unzip this compacted file to a directory. Then import it in IDEA. Since all libraries except java runtime have been contained in the project, you don't need to find them elsewhere. If IDEA cannot identify them, please fix these dependencies with those in the directory "lib".

## 3.3 Important files

The followings are some important files and their brief introductions. More details of their usages are discussed later in the next section.

### 3.3.1 MOP.java

This is a delegation file for the original "DoubleProblem" class in jmetal. The configuration for each test problem is defined in this file. With this file, you can construct test problems in two ways:

**Construct a single problem** Directly create an instance through this file with the following construction method:

MOP(String funName, int funNo, int M)

Here, *funName* is the name of the benchmark suite, *funNo* is the index of the problem in the suite, and *M* is the dimension of the objective space. For example, new MOP("WFG",1,15) will return a 15-objective WFG1 problem, while new MOP("MINUS\_DTLZ",3,3) will return a 3-objective DTLZ3<sup>-1</sup> problem.

**Construct a test suite** Call the static method "createManyObjectiveBenchmark" with the name of the test suite and obtain a list of test problems. Currently available test suite are:

- "WFG"
- "DTLZ1-4"
- "DTLZ5-7"
- "MINUS\_DTLZ1-4"

Note that, in the obtained problem list, one test problem will be repeated five times with the number of objective functions being 3, 5, 8, 10, 15, respectively.

### 3.3.2 Tools.java and StateTools.java

They are two utility files of this project. They mainly do the following things:

- Calculate aggregation function value for decomposition based algorithm
- Read weight vectors from files
- Do some statistical tests such as Wilcoxon test and Friedman test

Please keep these files unchanged unless you want to modify the above functions.

### 3.3.3 SimpleTest.java

This file is used to start an algorithm in a single run. This is usually useful when debugging the algorithms.

### 3.3.4 RunExperiment.java

With this file, you can run an experiment automatically. This is usually used when conducting a complete experiment.

## 3.4 CalculateIGD.java

After conducting an experiment and obtaining the final non-dominated solutions, use this file to calculate the IGD values. (Note that, you don't have to calculate the IGD values truly since all required data have been contained in the project.)

## 3.5 RunStatistics.java

This file is used to generate an excel file to present the experimental results.

## 4 Test the Algorithm

This section discusses how to run an algorithm in a single run.

The file SimpleTest.java contains a static main method so that you can launch this file directly. At the beginning of this main method, we should firstly define the test problems to be tested. An efficient way to choose the test problems is to obtain a list of problems with the MOP.java. Some related codes have already been implemented and all you need to do is to comment or uncomment the lines to choose a test suite.

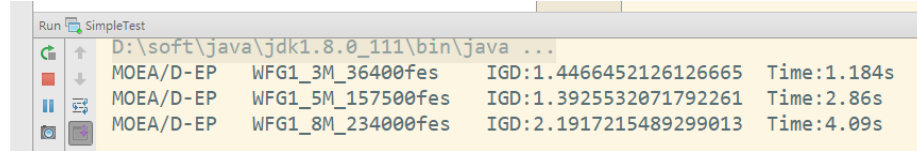
```
public class SimpleTest {
    public static void main(String[] args) throws Exception {
        List<MOP> plist = new ArrayList<>();
        plist.addAll(MOP.createManyObjectiveBenchmark( funName: "WFG"));
        // plist.addAll(MOP.createManyObjectiveBenchmark("DTLZ1-4"));
        // plist.addAll(MOP.createManyObjectiveBenchmark("DTLZ5-7"));
        // plist.addAll(MOP.createManyObjectiveBenchmark("MINUS_DTLZ1-4"));
    }
}
```

As shown in the above figure, the WFG test suite is chosen to be tested. You can uncomment the following three lines to enable the DTLZ and Minus DTLZ test suites.

When problems are obtained, they are merged into a list named *plist*. Then, a loop is started to execute an algorithm on each problem in this list. In this project, two algorithms including MOEA/D-EP and NSGAIII-EP are available. Both of these algorithms have a construction method accepting an MOP instance as a parameter. Examples of creating instances of these algorithms are shown in the following figure.

```
//      for (MOP p : plist) {
//          NSGA3_EP alg = new NSGA3_EP(p);
//          MOEA_EP alg = new MOEA_EP(p);
```

The above two steps are all you need to do. The following codes will start the algorithm, record the running time, calculate and output IGD values.



Algorithm	Test Problem	IGD	Time
MOEA/D-EP	WFG1_3M_36400fes	IGD:1.4466452126126665	Time:1.184s
MOEA/D-EP	WFG1_5M_157500fes	IGD:1.3925532071792261	Time:2.86s
MOEA/D-EP	WFG1_8M_234000fes	IGD:2.1917215489299013	Time:4.09s

## 5 Run an Experiment

Here we show how to run an experiment with runExperiment.java file.

### 5.1 Choose a test suite

runExperiment.java has a static main method so you can launch it. The first step is to choose the test problems, which is the same as described above. The following figure shows an example of choosing the DTLZ1-4 test problems.

```
27 public static void main(String[] args) throws Exception {
28     int maxRuns = 20;
29     List<MOP> plist = new ArrayList<>();
30     //      plist.addALL(MOP.createManyObjectiveBenchmark("WFG"));
31     plist.addALL(MOP.createManyObjectiveBenchmark( funName: "DTLZ1-4"));
32     //      plist.addALL(MOP.createManyObjectiveBenchmark("DTLZ5-7"));
33     //      plist.addALL(MOP.createManyObjectiveBenchmark("MINUS_DTLZ1-4"));
```

After that, each of these problems will be tested for 20 times. If you want to change the number of independent runs, set the variable *maxRuns* to another integer. These independent runs are executed parallelly with all available CPU cores. Therefore, it is recommended to close other applications before running the experiment.

## 5.2 Choose an algorithm

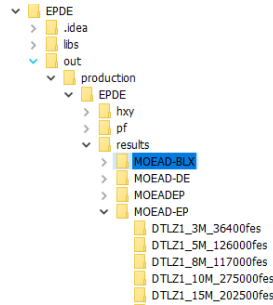
The first thing of choosing an algorithm is to give a name to the algorithm. This can be done by assigning a string to the variable *alg\_name*.

Then, in the try...catch... body, create an instance from MOEAD\_EP or NSGA3\_EP and assign it to the variable *alg*. When an exception is detected, the algorithm will be rerun automatically. The figure below shows an example of running MOEA/D-EP.

```
IntStream.range(1, maxRuns + 1).parallel().forEach((int runNo) -> {
    for (MOP p : plist) {
        String alg_name = "MOEAD_EP";
        Algorithm alg = null;
        boolean isSuccess;
        do {
            try {
                isSuccess = true;
                alg = new MOEAD_EP(p);
                alg.run();
            } catch (Exception e) {
                isSuccess = false;
                System.out.println("!!!error!!!");
            }
        } while (!isSuccess);
        save(alg, experiment_dir_path, alg_name, p, runNo);
    }
});
```

## 5.3 Output the experimental results

The results produced by the above experiments contain all the non-dominated solutions obtained on each test problem. All these results can be found in a directory "results" in the "out" directory. Detailed file structures are illustrated in the figure below.



In this example, all results are classified into some directories named by the algorithms. For each algorithm, there exist an unique directory stores related

results for a certain test problem. For instance, “DTLZ1\_3M\_36400fes” means the results obtained on 3-objective DTLZ1 within a number of 36400 function evaluations. Files contained in this directory are shown in the following figure.

Name	Size Auto	Modified
..		07.12.2017 11:27:26
Delta_p.txt	471 Byte(s)	19.09.2017 07:29:38
FUN1.tsv	5 kB	16.09.2017 16:28:40
FUN10.tsv	5 kB	16.09.2017 16:21:03
FUN11.tsv	5 kB	16.09.2017 15:25:46
FUN12.tsv	5 kB	16.09.2017 15:33:36
FUN13.tsv	5 kB	16.09.2017 15:37:59
FUN14.tsv	5 kB	16.09.2017 16:05:44
FUN15.tsv	5 kB	16.09.2017 16:13:24
FUN16.tsv	5 kB	16.09.2017 16:21:02
FUN17.tsv	6 kB	16.09.2017 16:28:40
FUN18.tsv	3 kB	16.09.2017 16:36:26
FUN19.tsv	5 kB	16.09.2017 16:44:11
FUN2.tsv	5 kB	16.09.2017 16:36:24
FUN20.tsv	4 kB	16.09.2017 16:51:49
FUN3.tsv	5 kB	16.09.2017 16:44:09
FUN4.tsv	5 kB	16.09.2017 16:51:47
FUN5.tsv	5 kB	16.09.2017 15:57:58
FUN6.tsv	5 kB	16.09.2017 15:25:45
FUN7.tsv	5 kB	16.09.2017 15:33:35
FUN8.tsv	4 kB	16.09.2017 16:05:44
FUN9.tsv	4 kB	16.09.2017 16:13:26
HV0905.txt	471 Byte(s)	17.09.2017 13:54:52
IGD.txt	478 Byte(s)	17.09.2017 20:27:58

Here, *FUN1.tsv* contains all non-dominated solutions obtained in the first run. Each line of this file contains the M objective values. *Delta\_p.txt*, *HV0905.txt*, and *IGD.txt* are files containing the corresponding indicator values. For instance, *IGD.txt* has 20 lines and the i-th line contains the IGD value obtained in the i-th run. (How to calculate these indicator values will be discussed later).

## 5.4 Calculate IGD values

The IGD indicator values for a given algorithm can be obtained using CalculateIGD.java. Specially, in the main method, set the variable *algName* to the name of the algorithm. For example, in the above instance, a directory named “MOEAD-EP” is obtained after running the experiment. Then, we can set *algName* to “MOEAD-EP” and run CalculateIGD.java. It will produce a file named “IGD.txt” for each test problem.

## 5.5 Output the results

When we get the results for more than one algorithms, we can output them to an excel file. To do this, write all the algorithm names to the variable *alg\_dir\_list* in the main method of RunStatistics.java. The figure below gives an example.

```

public static void main(String[] args) throws Exception {
    List<String> alg_dir_list = new ArrayList<>();

    alg_dir_list.add("NSGA3-EP");
    alg_dir_list.add("NSGA3");
    alg_dir_list.add("MOEAD-EP");
    alg_dir_list.add("MOEAD-SBX");
    alg_dir_list.add("MOEAD-BLX");
    alg_dir_list.add("MOEAD-DE");
    alg_dir_list.add("MOEAD-HOP");
}

```

Here, five algorithms are added to the comparative study. Run RunStatistics.java and an excel file named “report.xls” is generated in the “results” directory. This report looks like:

	A	B	C	D	E	F	G
	Problem	M	MOEAD-EP	MOEAD-SBX	MOEAD-BLX	MOEAD-DE	MOEAD-HOP
1	WFG1	3	4.920e-01	5.823e-01-	4.951e-01=	4.702e-01+	4.717e-01+
2		5	7.305e-01	8.323e-01-	6.044e-01+	4.184e-01+	4.450e-01+
3		8	9.298e-01	7.973e-01+	8.401e-01+	6.386e-01+	7.048e-01+
4		10	9.394e-01	7.011e-01+	8.405e-01+	7.225e-01+	8.914e-01+
5		15	9.114e-01	4.571e-01+	7.638e-01+	8.723e-01+	9.111e-01=
6	WFG2	3	8.934e-01	7.814e-01+	8.799e-01+	8.582e-01+	8.648e-01+
7		5	9.663e-01	8.008e-01+	8.165e-01+	9.494e-01+	9.439e-01+
8		8	9.573e-01	7.773e-01+	8.054e-01+	9.295e-01+	9.313e-01+
9		10	9.635e-01	7.985e-01+	8.205e-01+	9.482e-01+	9.477e-01+
0		15	9.593e-01	7.871e-01+	8.234e-01+	9.400e-01+	9.510e-01+
1	WFG3	3	7.246e-01	6.878e-01+	7.308e-01-	6.906e-01+	6.871e-01+
2		5	6.486e-01	6.220e-01+	6.221e-01+	6.022e-01+	6.117e-01+
3		8	5.027e-01	4.207e-01+	3.850e-01+	4.917e-01=	5.034e-01=
4		10	3.056e-01	3.020e-01+	3.050e-01=	4.384e-01-	4.400e-01-
5		15	2.475e-01	2.425e-01+	2.443e-01+	2.444e-01+	2.454e-01+

In this excel file, “+”, “-”, and “=” denotes the target algorithm (the first one in *alg\_dir\_list*) is better than, worse than, or competitive with the peer algorithm, respectively, at a 0.05 significance level. It should be pointed out that, the Wilcoxon test in this java project is implemented by ourselves and its results have some slight differences from those in the paper.

One other thing to note is only the  $\Delta_p$  indicator is available on the Minus DTLZ problems. So, when exporting the results on these problems, one should remove IGD and HV from the available indicator metrics list. This can be done by change the *use\_indicator* variable. The following figure gives an example of choosing  $\Delta_p$  as the only one indicator.



```
public class RunStatistics {  
    String output_dir_path = Tools.class.getResource(".").getPath() + "/results/";  
    // List<String> use_indicator = Arrays.asList("IGD");  
    // List<String> use_indicator = Arrays.asList("HV0905", "IGD", "Delta_p");  
    List<String> use_indicator = Arrays.asList("Delta_p");  
}
```

---

If you encounter any questions in using this project, please contact with Mr. He (email: [hxyokokok@foxmail.com](mailto:hxyokokok@foxmail.com)). We will try our best to fix them.