Chapter 4

Experiment Plans

4.1 Empirical evidence

As discussed in previous chapters, we theoretically prove that it is sufficiently safe to rewrite these algorithms in terms of the unified framework (see Chapter 2) with the generic dictionary (see Chapter 3). However, **theoretical evidence alone is not enough, and empirical evidence is also essential**. Therefore, this chapter will introduce our experiment plans to provide empirical evidence.

We designed the experiment as follows:

(1) **Research question**: We want to know if the unified framework we designed for these seven algorithms could adequately replace their original framework. Specifically, we want to know whether the same algorithm in two different representations behaves identically on solving a set of questions.

(2) **Define main variables**: The algorithm representation shall be the independent variable, and the performance of algorithms shall be the dependent variable.

(3) **Our hypothesis**: The unified framework we designed can correctly cover these seven algorithms. Specifically, for these seven algorithms, the performances of each algorithm between written in the unified framework and written in its original framework are the same on solving a set of questions.

(4) **Experimental treatment**: We will implement both representations of each algorithm in Python. Each algorithm in both frameworks will solve the same set of optimization problems.

(5) **Measure dependent variable**: The performances of each algorithm in both frameworks will be measured by the number of problem evaluations [13]. The comparison between each algorithm's two representations will be measured by a statistical test method.

Therefore, we need a tool —IOHprofiler [7] —that can access enough optimization problems for testing algorithms, and allows the number of problem evaluations to be the criteria of algorithms' performance. Furthermore, considering these seven algorithms are created for solving continuous optimization problems [14–18, 31, 32], we will select the BBOB problem set for benchmarking algorithms. Specifically, the BBOB set merged into the IOHprofiler framework contains 24 different single-objective continuous optimization problems searching for a minimal solution [9].

The tool IOHprofiler [7, 30] consists of an experimental part and a post-processing part. The experimental part is used to generate time-series running data, and the post-processing part will quantify the algorithm performance by analyzing these generated data. **In the experimental part**, we will manually implement each algorithm in both representations. For example, we will define the stop condition, the dimension of problems, the number of instances of each problem, and the number of runs of each instance [1]. **In the post-processing part**, we will compare the performance of each algorithm in the unified framework and its original framework by observing their ERT plots obtained from IOHanalyzer [2]. Here, because we are mostly interested in how many evaluations the algorithm will take when faced with different target values, the ERT plot visualizes how many evaluations each algorithm in each framework will take to reach a given target value, where the x-axis denotes various target values whose minimum value is 10^{-8}, and the y-axis denotes the number of evaluations called Expected Running Time (ERT) whose maximum number is $n \times 10^4$. n here is the dimension of problems.

The IOHanalyzer also provides R programming interfaces in which the area under the ECDF curve for each algorithm in each framework on solving each problem will be collected for further analysis. Here, ECDF for each problem is an aggregated Empirical Cumulative Distribution over a set of target values. The set of target values obtained from IOHanalyzer is $\{10^i \mid i = 2, 1.8, 1.6, ..., -8\}$. Furthermore, repeating experiments multiple times is essential to ensure the experimental results are scientific. Therefore, we conclude how we set up and use the IOHprofiler environment in Table 4.1.

[1] The instance of each continuous optimization problem is its variants by multiplicative shift and/or additive shit. Please find more details in [7].

[2] https://iohanalyzer.liacs.nl/

Table 4.1: Setups and usages in IOHproflier.

Environment	Setups	Usages
IOHexperimenter	· minimum target value: 10^{-8}.	· generate running data of each problem over multiple instances and multiple runs.
	· maximum number of evaluations: $n \times 10^4$.	
	· dimensions of problems: $n = 5, 20$.	
	· the number of instances for each problem: 5.	
	· the number of runs for each instance: 5.	
IOHanalyzer	· web-based GUI	· observe ERT plots.
	· R programming interfaces.	· calculate AUCs of ECDFs.

As shown in Table 4.1, when measuring experimental results, ERT plots will describe differences of each algorithm performance in the unified framework and its original framework in the view of quality. We also need to measure the differences in the view of quantity. The AUC value of ECDF curve will measure each algorithm's performance in each framework. Precisely, each algorithm in each framework will have 24 AUC values throughout 24 optimization problems obtained in IOHexperimenter. Considering we are interested in differences between each algorithm performance in our designed unified framework and its original framework, also, both of them (the algorithm in our designed unified framework and the same algorithm in its original framework) will solve the same 24 problems, a kind of paired sample test statistical procedure will be preferred.

Lastly, as mentioned in previous paragraphs, only two measurement indicators (ERT and AUC of ECDF) will be used for analyzing experimental results. The ERT plot will visually display how the number of evaluations increases (in the y-axis) as the target value decreases (in the x-axis). In the ERT plot, one line denotes the performance of one algorithm in one framework over one dimension on one problem. The AUC values of ECDF will quantify the probability that each algorithm in each framework successfully reaches the target value. Each AUC value denotes such a probability for one algorithm in one framework over one dimension on one problem. Here, we want to clarify that the

IOHprofiler environment has its own mechanism to deal with multiple instances and runs. Simply speaking, this mechanism is a kind of average calculation, but more accurate details can be found in [7, 30].

4.2 Summary

In this chapter, we introduced the purpose of doing such practical experiments to give empirical evidence on whether these seven algorithms can safely be rewritten in our designed unified framework.

For achieving such experiments, we will test the performance of each algorithm in each framework with the help of the IOHprofiler benchmarking tool. Although the IOHprofiler provides various statistical ways to measure the performance of algorithms, we will focus on two measurements: ERT plots in the view of quality and AUC values of ECDF in the view of quantity.

Moreover, the AUC values will be further analyzed by a kind of paired sample test statistical method that will be determined in actual experiments (see Chapter 5).