Chapter 4

~~Experiment Plans~~ Experimental Setup of Benchmark Study

4.1 Motivation for experiments

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 general 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.

here, you only mention the comparison between the original and your implementation. But our motivation is also to understand how well (or not) the algorithms perform on standardized benchmark problems.

Table 4.1: Setups and usages in IOHproflier.

Environment	Setups	Usages
	• minimum target value: 10^{-8}	

(5) **Measure dependent variable**: The performances of each algorithm in both frameworks will be measured by the number of problem evaluations [14]. The comparison between each algorithm's two representations will be measured by a statistical test method.

Therefore, we need a tool —IOHprofiler [8] —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 [15–19, 32, 33], 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 [10].

The tool IOHprofiler [8, 31] 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 $\{10^i \mid i = 2, 1.8, 1.6, \ldots, -8\}$. 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 [8].

[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

lems. in search and in objective spa

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 [8, 31].

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).

and AUC

[redacted] ECDF should be explained.

please check the formulation with someone.

between syncG and asyncG orig_BA algorithm's performance is significant. Such same observation is also found in orig_GOA algorithm when dimension n is 20.

Considering the difference is not significant in other cases, further analysis is needed to make the role of $\mathbf{x}_g$ explicit in unsignificant cases (orig_MBO, orig_BOA, orig_PSO) compared to significant cases (orig_BA, orig_GOA). From Table 5.3 that lists the way of each algorithm using $\mathbf{x}_g$ in ORIGINAL framework and UNIOA framework, it is observed that the effect of $\mathbf{x}_g$ is different in these algorithms, although $\mathbf{x}_g$ affects the quality of optimization in all of these algorithms. For example, $\mathbf{x}_g$ is directly added to objective solution $\mathbf{x}_i$ in BA and GOA, but in MBO, BOA and PSO, $\mathbf{x}_g$ is scaling down by multiplying a very small decimal (BOA), or by subtracting a larger value (PSO), or by reducing the probability of using $\mathbf{x}_g$ (MBO).

Therefore, ~~it is preferred to~~ we conclude that **syncG and asyncG algorithms have different performance on average for a random optimization BBOB function in ORIGINAL framework, no matter which dimension it is in.**

Please add a conclusion: which version do you decide to use (and why)? for the rest of the paper.

...continued

Case	p-value (n=5)	p-value (n=20)
orig_MBO_asyncE_syncG_or_asyncG	0.19854279368666194	0.07411601304083099
orig_BOA_asyncE_syncG_or_asyncG	0.8192020334011836	0.2773986245500173
orig_PSO_asyncE_syncG_or_asyncG	0.44045294529422474	0.964388671614557

Table 5.3: How each algorithm uses $\mathbf{x}_g$ in ORIGINAL framework and UNIOA framework.

H_0	Algorithm	$\mathbf{x}_g$ role in ORIGINAL	$\mathbf{x}_g$ role in UNIOA
reject	BA	Eq.2.5, Eq.2.6	Eq.3.15
reject	GOA	Eq.2.10	Eq.3.19
-	CSA	NO USE $\mathbf{x}_g$	
-	MFO	NO USE $\mathbf{x}_g$	
not reject	MBO	Step (6)	Eq.3.30
not reject	BOA	Eq.2.28	Eq.3.34
not reject	PSO	Eq.2.34	Eq.3.36

5.2.2 Conclusion

According to the experimental results in Section 5.2, we can conclude that the different effects from asyncE and syncE could be ignored, but whether the way of calculating $\mathbf{x}_g$ is synchronous or asynchronous much likely has negative impacts on measuring the performance of algorithms. In other words, when the performance of UNIOA is the same as the performance of ORIGINAL, it is much likely because different ways of calculating $\mathbf{x}_g$ force their performance to behave same, but not because the unified framework itself does not affect the algorithm performance.

Therefore, as shown in Table 5.4, we modify asyncG to syncG, but not change their ways of evaluation. Here, considering the way of calculating $\mathbf{x}_g$ in UNIOA is synchronous, we choose to change asyncG to syncG, but not change syncG to asyncG. This is because we want to avoid misleading our conclusions about whether UNIOA performs same

as ORIGINAL. For example, when the performance of UNIOA is different from the performance of ORIGINAL, we hope the reason is the fault of our unified framework, but not the way of calculating $\mathbf{x}_g$.

? I don't understand this sentence.

Table 5.4: hfngnhmgh.

Algorithm	Evaluation		$\mathbf{x}_g$	
	in original code	in this work	in original code	in this work
orig_BA	asyncE	asyncE	asyncG	syncG
orig_GOA	syncE	syncE	syncG	syncG
orig_CSA	syncE	syncE	no use	no use
orig_MFO	asyncE	asyncE	no use	no use
orig_MBO	syncE	syncE	syncG	syncG
orig_BOA	asyncE	asyncE	asyncG	syncG
orig_PSO	asyncE	asyncE	asyncG	syncG

Comparing UNIOA to the original implementations

5.3 Experiments for verifying the unified framework

5.3.1 Results

this long sentence → split in 2 sentences

In Section 5.2, we eliminate side effects from $\mathbf{x}_g$ calculation method when reproducing these algorithms in ORIGINAL framework, subsequently, it is much safer to discuss the difference between ORIGINAL and UNIOA.

From Figure 5.3 in which y-axis is each algorithm whose framework is ORIGINAL or UNIOA and its corresponding AUC values obtained in IOHanalyzer are shown on x-axis, it is observed that the two distributions of 24 AUC values for ORIGINAL algorithm and for UNIOA algorithm are the same. Even the outliers from ORIGINAL algorithm and from UNIOA algorithm are also significantly near to each other. Same observations can be found when dimension n is 5 and when dimension n is 20.

Figure 5.
framewor

Meanwhil
algorithm
n is 5, al
from ORI
other wor
significan

Furthermo
is significa
UNIOA_MI
better than
AUC values
values in 2

or UNIOA had when comparing their own AUC value throughout 24 BBOB functions.

Dimension n	Case	p-value	orig win	UNIOA win
$n = 5$	BA	0.11276741428797125	7	17
	GOA	0.0804264513256041	15	9
	CSA	0.05933346675499405	9	15
	MFO	0.17931823604537578	11	13
	MBO	0.37577150825113037	14	10
	BOA	0.8638867905449266	12	12
$n = 20$	PSO	0.006090717662464104	8	16
	BA	0.6558614582873129	3	21
	GOA	0.06569860480594888	3	21
	CSA	0.7860401269462913	12	12
	MFO	0.000191118942510799	4	20
	MBO	0.14097231038635524	15	9
	BOA	0.8327296566664774	6	18
	PSO	0.6064080161085506	6	18

5.3.2 Conclusion

According to the experimental results in Section 5.3, we can conclude that at least in these seven algorithms, the ORIGINAL framework can be safely replaced by our unified UNIOA framework, in the point of actual algorithm performance.

with respect to

For example, in Figure 5.4 showing fix-target curves, the purple line denotes the MFO in ORIGINAL, and the orange line denotes the MFO in UNIOA. We can find two lines are close to each other in most of 24 functions, which means the MFO algorithm performs same in two different frameworks when solving most of 24 optimization problems. Sometimes,

the MFO in ORIGINAL needs more evaluations to reach the same target value as the MFO in UNIOA, when solving F1. Sometimes, the MFO in UNIOA can obtain much better optimization results than the MFO in ORIGINAL, when solving F2, F7, F17, F21. However, the MFO in ORIGINAL performs slightly better than the MFO in UNIOA when solving F8, F9, F15.

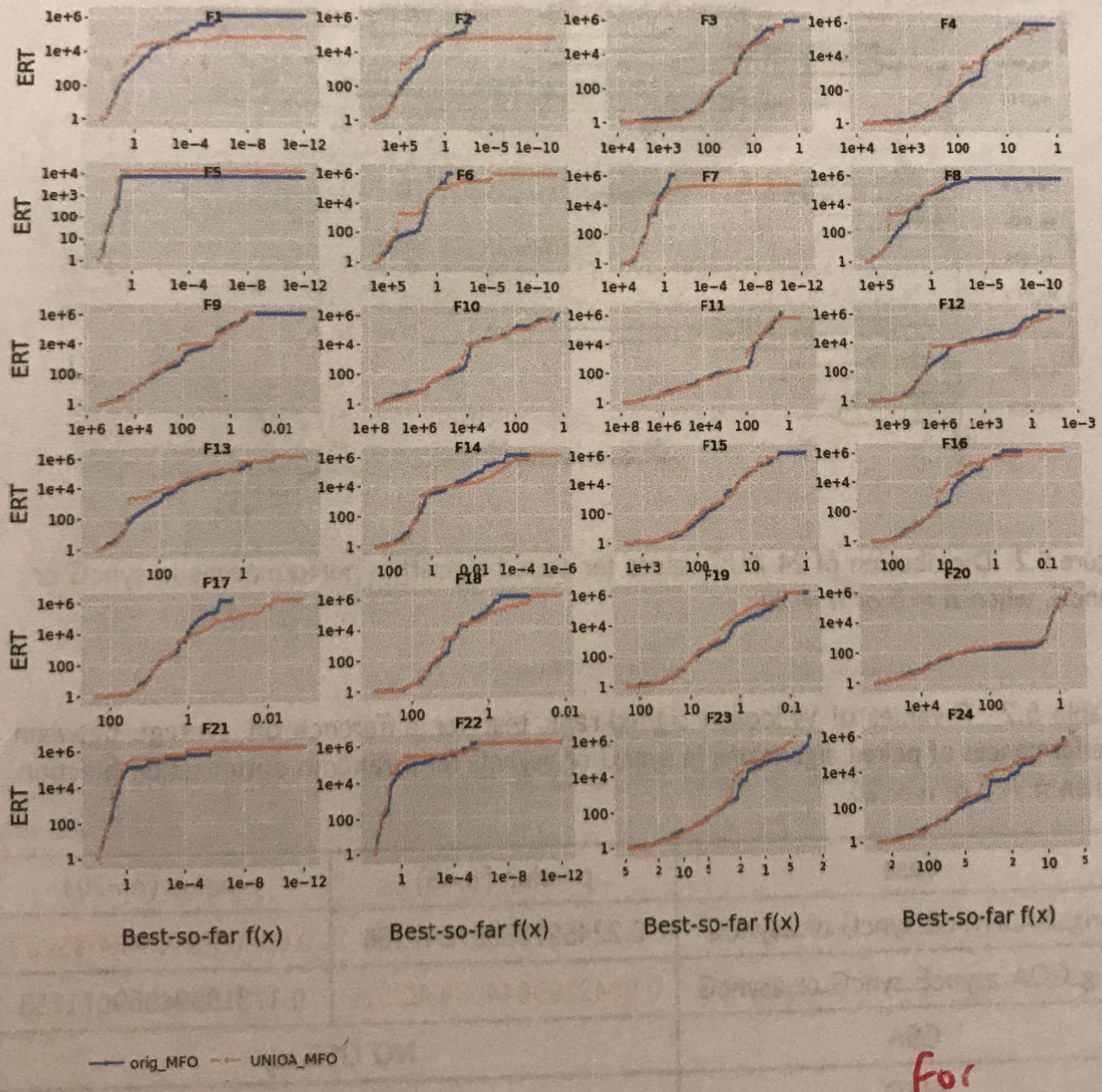

Figure 5.4: ERT values of the MFO in UNIOA and in ORIGINAL, when $n = 5$.

For

Specify here how many runs you have done.

Chapter 5

Experimental Results

5.1 Introduction

In Chapter 4, we point out the need to further demonstrate the reliability of the unified framework UNIOA with its general terminologies through actual experiments. We also briefly describe the experimental environment with its settings shown in Table 4.1. In this chapter, firstly, we detail how many kinds of experiments are displayed to achieve the purpose of experiments. Next, we answer the question mentioned in Section 4.2 about what kind of statistics test method will be used in this work. Lastly, we display the experimental results and give discussions.

Because we are interested in how close the performances of one algorithm are in its original framework and our unified framework, the framework structure shall be the only aspect affecting the algorithm performance and any other aspects with side effects on algorithms' performance shall be eliminated. Therefore, **the first group of experiments is to test whether syncE/asyncE and syncG/asyncG will significantly affect the algorithm performance, and significant effects shall be eliminated** [1]. The experimental results of this group are displayed in Section 5.2.

After eliminating some aspects with side effects, **the second group of experiments is to compare the performances of each algorithm in its original framework**

[1] In a variety of experimental attempts at measuring the performance of algorithms, we find these two aspects (syncE/asyncE: the evaluation method is synchronous or asynchronous. syncG/asyncG: the method of calculating the global best individual is synchronous or asynchronous.) that might significantly affect the algorithm performance. Therefore, we do experiments to make sure the influence of them.

and our unified framework. The experimental results of this group are displayed in Section 5.3.

move to Section 4 ?

Then, for providing convincing discussion, as discussed in Chapter 4, instead of a common two-sample test, a paired test shall be preferred. Moreover, rather than the popular paired t-test, we prefer the Wilcoxon signed-rank test with a confidence level of 95% because samples in our case are not normally distributed. ***The null hypothesis H_0 in our experiments is the difference between two algorithms' performances is on average zero for a random BBOB optimization problem.*** Here, the two algorithms mean two versions of one algorithm. For example, they could be (1) syncE bat-algorithm and asyncE bat-algorithm. (2) syncG bat-algorithm and asyncG bat-algorithm. (3) bat-algorithm in its original framework and bat-algorithm in our unified framework. Because we studied seven different algorithms, such experiments will be repeated seven times.

After observing the performances of each algorithm in its original framework and our unified framework, we are interested in the actual application of this framework. In Section 5.4, we achieve our designed framework into a Python package named UNIOA in which various ideas of swarm-based optimization algorithms can be prototyped in the unified framework, evaluated with the help of the IOHprofiler environment, and compared with each other.

In the Section 5.5, we test our UNIOA package to compare these seven algorithms. Therefore, **the third group of experiments is to compare the performances of algorithms in our unified framework for giving an overview understanding of these algorithms**.

Lastly, to make the discussion in the following sections clearer and easier to understand, we use 'UNIOA' to denote the algorithm in the unified framework with unified terminologies and 'ORIGINAL' (abbr. orig) to denote the algorithm model in its original framework. The chapter is concluded in Section 5.6.

5.2 Experiments for avoiding side effects

5.2.1 Results

In the first group of experiments, we are interested in whether different ways of calculating the fitness and the global best individual will significantly impact the performance of algorithms. In many experimental attempts, we found that there are two ways to calculate fitness:

each case represents one comparison between one algorithm whose evaluation method is synchronous or is asynchronous. We find that all p values are larger than 5%. It means **the data obtained from syncE algorithm and asyncE algorithm does not reject the hull hypothesis, no matter which framework it belongs to and no matter which dimension it is in.**

→ what does this imply? Provide your interpretation/discussion of this finding.

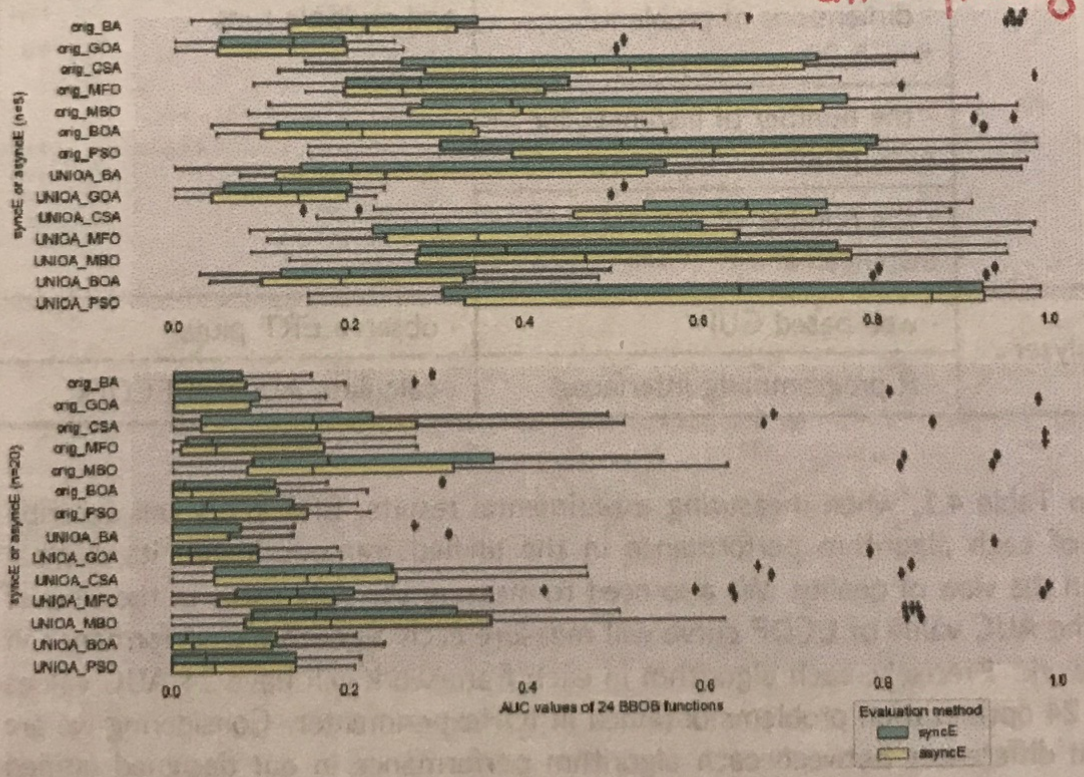

Figure 5.1: Distribution of 24 AUC values for paired algorithm performances in syncE or asyncE, when $n = 5$ or $n = 20$.

(top) (bottom)

between syncG and asyncG orig_BA algorithm's performance is significant. Such same observation is also found in orig_GOA algorithm when dimension n is 20.

Considering the difference is not significant in other cases, further analysis is needed to make the role of $\mathbf{x}_g$ explicit in unsignificant cases (orig_MBO, orig_BOA, orig_PSO) compared to significant cases (orig_BA, orig_GOA). From Table 5.3 that lists the way of each algorithm using $\mathbf{x}_g$ in ORIGINAL framework and UNIOA framework, it is observed that the effect of $\mathbf{x}_g$ is different in these algorithms, although $\mathbf{x}_g$ affects the quality of optimization in all of these algorithms. For example, $\mathbf{x}_g$ is directly added to objective solution $\mathbf{x}_i$ in BA and GOA, but in MBO, BOA and PSO, $\mathbf{x}_g$ is scaling down by multiplying a very small decimal (BOA), or by subtracting a larger value (PSO), or by reducing the probability of using $\mathbf{x}_g$ (MBO).

Therefore, it is preferred to conclude that **syncG and asyncG algorithms have different performance on average for a random optimization BBOB function in ORIGINAL framework, no matter which dimension it is in**.

Table 5.1: P-values of Wilcoxon signed-rank test for difference on average between performances of paired algorithms in syncE or asyncE for a random optimization function, when $n = 5$ or $n = 20$.

Case	p-value (n=5)	p-value (n=20)
orig_BA_syncE_or_asyncE_syncG	0.3313349065031326	0.22886900026455015
orig_GOA_syncE_or_asyncE_syncG	0.6261109162613098	0.6673653270475619
orig_CSA_syncE_or_asyncE	0.6475683676310555	0.8862414820514412
orig_MFO_syncE_or_asyncE	0.2530979089471155	0.8409995729722781
orig_MBO_syncE_or_asyncE_syncG	0.31731050786291415	0.17469182698689223
orig_BOA_syncE_or_asyncE_syncG	0.7750969621959847	0.2773986245500173
orig_PSO_syncE_or_asyncE_syncG	0.954431397113681	0.19449921074798193
UNIOA_BA_syncE_or_asyncE_syncG	0.24142655338204444	0.5968445170755943
UNIOA_GOA_syncE_or_asyncE_syncG	0.6465582592577419	0.6233436223982587
UNIOA_CSA_syncE_or_asyncE	0.9090113066460508	0.9430276454464167
UNIOA_MFO_syncE_or_asyncE	0.6475683676310555	0.38327738184229543
UNIOA_MBO_syncE_or_asyncE_syncG	0.265156633625956	0.24697273165906564
UNIOA_BOA_syncE_or_asyncE_syncG	0.09749059620220792	0.629275096131297
UNIOA_PSO_syncE_or_asyncE_syncG	0.954431397113681	0.4655179892460418

From Figure 5.2 in which y-axis is each algorithm whose $\mathbf{x}_g$ calculation method could be synchronous or asynchronous and its corresponding AUC values obtained in IOHanalyzer are shown on x-axis, it is observed that the two distributions of 24 AUC values for syncG algorithm and for asyncG algorithm are varying from algorithms. For example, the difference of paired performances between syncG BA and asyncG BA is significant when dimension n is 5 and when dimension n is 20, however, this kind of difference shown in Figure 5.2 is not significant in other algorithms. From Table 5.2 in which each case represents one comparison between each algorithm whose $\mathbf{x}_g$ calculation method is synchronous or is asynchronous, it is observed that when dimension n is 5, the data obtained from orig_GOA algorithm rejects the hull hypothesis which means the difference