Evolving Benchmark Functions Using Kruskal-Wallis Test *

Yang Lou[1,a], Shiu Yin Yuen[1,b], and Guanrong Chen[1,c]

[1] *Department of Electronic Engineering, City University of Hong Kong*
[a] felix.lou@my.cityu.edu.hk
[b] kelviny@cityu.edu.hk
[c] eegchen@cityu.edu.hk

Abstract

Evolutionary algorithms are cost-effective for solving real-world optimization problems, such as NP-hard and black-box problems. Before an evolutionary algorithm can be put into real-world applications, it is desirable that the algorithm was tested on a number of benchmark problems. On the other hand, performance measure on benchmarks can reflect if the benchmark suite is representative. In this paper, benchmarks are generated based on the performance comparison among a set of established algorithms. For each algorithm, its uniquely easy (or uniquely difficult) problem instances can be generated by an evolutionary algorithm. The unique difficulty nature of a problem instance to an algorithm is ensured by the Kruskal-Wallis H-test, assisted by a hierarchical fitness assignment method. Experimental results show that an algorithm performs the best (worst) consistently on its uniquely easy (difficult) problem. The testing results are repeatable. Some possible applications of this work include: 1) to compose an alternative benchmark suite; 2) to give a novel method for accessing novel algorithms; and 3) to generate a set of meaningful training and testing problems for evolutionary algorithm selectors and portfolios.

1 Introduction

Benchmark test problems are important for performance measure of evolutionary algorithms (EAs). Usually, users can decide whether an EA should be applied, or which EA should be applied, to real-world optimization problems based on the historical performance in comprehensive benchmark testing studies. Strengths and weaknesses of EAs are empirically studied by testing on many problem instances [1, 2, 3], since theoretical investigations are difficult if not possible. A good benchmark suite is required to be representative of real-world problems, which is also a difficult task for testing. Practically, benchmark suites are proposed to partially cover certain target portions of the entire real-world problem domain. For example, [4] gives a set of multi-modal benchmark problems, to mimic the intrinsic multi-modality in engineering optimization problems.

Generally, there are four categories of benchmark problems that are frequently used: 1) annually proposed competition benchmarks, e.g. the black box optimization benchmarking (BBOB) [5]; 2) well-known benchmark problems used in influential articles, e.g. [6]; 3) tunable benchmark generators, e.g. the max-set of Gaussian (MSG) generator [7]; and 4) real-world problems [8]. The first three categories are simulation problems. Though successfully working on real-world applications is the ultimate goal for EAs to achieve, artificial simulation data play an important role in understanding the real-world data, of which the ground truth is unknown [9]. The no free lunch (NFL) theorem [10] suggests that no algorithm is better than any other when all possible black box type problems are equally likely to occur. The NFL theorem also warns us about the risk in generalization of EA performance, e.g. the risk that an EA is efficient on human-designed

*Cite paper: Yang Lou, Shiu Yin Yuen, and Guanrong Chen, "Evolving Benchmark Functions Using Kruskal-Wallis Test," In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (*GECCO* '18), Kyoto, Japan, July 2018, pp. 1337–1341. Doi: 10.1145/3205651.3208257, 2018.

benchmarks, but poor on real-world problems. Since different problem categories are not mutually exclusive, and one cannot investigate *all problems*, there is always a chance to find a subset of problems on which one can declare an algorithm performs significantly better than another.

Tunable generators are flexible and are able to cover a wide range of problems, but parameter tuning is a problem that hinders the wide use of tunable generators. Random parameters are not encouraged, since the resulting problems lack diversity [11]. Generating benchmark instances using evolutionary computation [12, 13, 14, 15, 16] allows one to obtain problem instances with required features.

A single EA is employed in [13] for generating combinational benchmark problems that are more difficult than commonly used test suites. The difficulty of a problem is measured by the single-run search effort (i.e., the number of search operations) for the algorithm to obtain a result with the user-defined precision.

Genetic programming is used to evolve discretized problem instances in [15], where the performance of two EAs are compared. Novel benchmark instances are generated by maximizing the performance difference between the two EAs.

Performance comparison among multiple (i.e., three or more) algorithms is considered a multi-objective optimization problem in [16], with each objective representing a pairwise comparison as in [15]. In [15], results are averaged from five independent runs to filter out randomness.

EA performance is stochastic, rather than deterministic. Previous benchmarking generation works scarcely consider algorithm performance as a statistical problem, however.

The hierarchical-fitness-based evolving benchmark generator (HFEBG) proposed in [17, 21] gives an approach to generate uniquely easy (UE) benchmark suites for EAs. The difficulty of a problem is defined by performance, and a UE problem with respect to an EA is defined by performance comparison among the employed EAs. Thus, for any algorithm A_i ($i = 1, \ldots, N$), a problem is UE to A_i if and only if A_i outperforms any other algorithm A_j ($j = 1, \ldots, N, j \neq i$) in solving this problem, and their performance difference is statistically significant. Similarly, a uniquely difficult (UD) problem for A_i means that A_i performs significantly worse than any other algorithm A_j ($j = 1, \ldots, N, j \neq i$) in solving this problem.

A set of established algorithms are employed in HFEBG. For each algorithm, a UE problem instance is generated by evolution. The resulting instances consist of a novel benchmark test suite. Each EA outperforms any other EA statistically significantly on its UE problem. Therefore, a relatively unbiased benchmark suite is composed. The statistical significance is guaranteed by the Mann-Whitney U-test [18].

In this paper, we propose an alternative way of statistical guarantee in HFEBG, by using the Kruskal-Wallis H-test [19] instead of the Mann-Whitney U-test. The H-test offers a non-parametric statistical test multiple-sample-wise, while the U-test is for pairwise comparison. The performance comparisons of multiple EAs are converted to the comparison of two EAs in [21], i.e., the target EA and the other EAs. The other EAs compose a metaheuristic portfolio, which represents the best performance of all the other EAs. If the target EA outperforms the portfolio on a problem instance, then this instance is considered a UE problem for the target EA. In this paper, since the performance of multiple EAs is compared directly using H-test, there is no need to compose a metaheuristic portfolio. The proposed method is named as HFEBG-H (hierarchical-fitness-based evolving benchmark generator with the Kruskal-Wallis H-test). Also, in [21], even if each pair of algorithms pass the U-test, it does not mean that the target EA is superior to the other algorithms as a whole. This mathematical inaccuracy is corrected by employing the Kruskal-Wallis H-test.

The rest of the paper is organized as follows: Section 2 reviews HFEBG and describes the details of the proposed work. In Section 3, simulation results are presented. Section 4 concludes the paper.

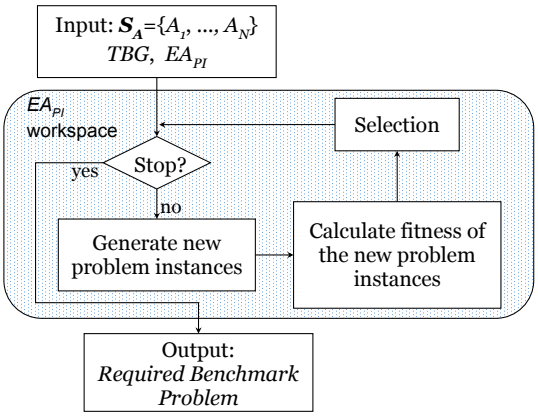

Figure 1: The flowchart of HFEBG generating a uniquely easy/difficult problem for an EA within an EA set.

2 The Benchmark Generator

2.1 General Framework

Without loss of generality, we consider optimization as minimization in this paper, so we will use the two terms interchangeably if no confusion would arise. We also use problem and problem instance interchangeably, since in this work a (benchmark) problem for an algorithm is also an instance in the tunable problem space.

The general HFEBG framework of evolutionary problem instance generator is shown in Fig. 1. The input includes: 1) A set of N algorithms, $\boldsymbol{S_A} = \{A_1, A_2, \ldots, A_N\}$ (a bold means a vector or matrix); 2) a tunable benchmark instance generator (TBG); and 3) an EA for evolving problem instances (EAPI).

The EAPI workspace is the main loop for generating problems. In this work, we employ differential evolution (DE) [20] as EAPI. The output of HFEBG is the required problem, i.e., a UE or UD problem.

HFEBG-H follows the framework of HFEBG and uses the same definition of UE/UD problems as defined in [17, 21]. However, the proposed HFEBG-H uses a different fitness assignment method, which is based on multiple comparison.

2.2 Hierarchical Fitness with H-test Results

Suppose algorithm A_i is the current target EA, meaning that the HFEBG-H is generating a UE/UD problem for A_i (re-denoted by A_T). The other algorithms in $\boldsymbol{S_A}$ are re-denoted by $\boldsymbol{B_A} = \boldsymbol{S_A} - A_T = \{B_1, \ldots, B_{N-1}\}$. $\boldsymbol{B_A}$ can be considered as a set of algorithms to be beaten by A_T (for the UE-A_T case), or to beat A_T (for the UD-A_T case).

The hierarchical fitness with three components is denoted by $fit = \{fit.h_1, fit.h_2, fit.h_3\}$, where $fit.h_1$ has the highest priority, and $fit.h_3$ has the lowest. In the following, we use the process of generating UE problems as an example to illustrate the assignment method of the hierarchical fitness with the Kruskal-Wallis H-test. Suppose a problem instance I_x is generated by the HFEBG-H. The hierarchical fitness of I_x is assigned based on the multiple performance comparison among algorithms. Each algorithm is given R independent runs to solve I_x, and then the results are collected to measure the relative difficulty of the problem instance I_x.

1) If the mean result obtained by A_T is better than all algorithms in $\boldsymbol{B_A}$ (meaning that A_T performs better than $\{B_1, \ldots, B_{N-1}\}$, without statistical guarantee), then $fit.h_1$ is assigned to indicate how much A_T is better than $\{B_1, \ldots, B_{N-1}\}$. The Kruskal-Wallis H-test and the

p-value correction of multiple comparison are used to calculate the p-values: $p_1 = H(A_T, B_1)$, $p_2 = H(A_T, B_2)$, ... , $p_{N-1} = H(A_T, B_{N-1})$, where $H(x, y)$ represents the corrected p-value between samples x and y within the context of multiple comparison, and $fit.h_1 = \max\{p_1, p_2,$ ... $, p_{N-1}\}$. Given a p-value p $(0 < p \leq 1)$, if $p \leq \alpha$, the existence of significant difference with a confidence level α is declared; otherwise, $p > \alpha$, meaning no significant difference. Therefore, by obtaining the maximum p-value among $p_1, p_2,$ ... $, p_{N-1}$, $fit.h_1$ represents the minimum performance difference between the target EA (A_T) and the best performing EA in $\boldsymbol{B_A}$. In this case, the other two components ($fit.h_2$ and $fit.h_3$) are assigned a value of empty (Φ).

2) If the mean result obtained by A_T is better than several EAs in $\boldsymbol{B_A}$ (e.g., A_T performs better than $\{B_1, B_3\}$ but worse than $\{B_2, B_4, B_5\}$), then $fit.h_2$ is assigned the negative number of EAs beaten by A_T, while the other two components of fit are assigned the value of empty (Φ). For example, A_T performs better than two algorithms $\{B_1, B_3\}$, then $fit.h_2 = -2$.

3) If A_T is worse than all algorithms in $\boldsymbol{B_A}$ (meaning that A_T obtains worse mean result than the mean result of any algorithm in $\{B_1,$ ... $, B_{N-1}\}$), then $fit.h_3$ is assigned the residual value between the result obtained by A_T and the worst performing EA in $\boldsymbol{B_A}$. Recall that the aim here is to generate a UE problem for A_T, but unfortunately A_T performs the worst on solving I_x, thus $fit.h_3$ represents how far A_T falls behind other EAs in terms of mean results. The other two components of fit are assigned the value of empty (Φ).

Table 1 gives an example of hierarchical fitness values, where fit_1 represents the best solution (problem instance), while fit_3 is the worst. The hierarchical comparison is performed as follows: 1) if two fitness values have different levels of non-empty components, then the one with a higher level non-empty component is better; and 2) if the two fit values have the same level of non-empty components, then the one with a lower value wins, i.e., a minimization problem within each component. For example, fit_1 is the best, because it has the highest level non-empty component; fit_4 is better than fit_2, because both have the same level non-empty component ($.h_2$), but fit_4 has a lower value; fit_3 is the worst since it has the lowest level non-empty component.

Within the EAPI workspace shown in Fig. 1, newly generated benchmark problems are assigned with hierarchical fitness, and then compared in the selection step. Therefore, the required (UE/UD) problem instances can be generated gradually by evolution.

Table 1: An example of hierarchical fitness values.

	$.h_1$	$.h_2$	$.h_3$
fit_1	0.05	Φ	Φ
fit_2	Φ	-2	Φ
fit_3	Φ	Φ	1.41
fit_4	Φ	-3	Φ

3 Experimental Studies

3.1 Experimental Settings

The max-set of Gaussian (MSG) generator [7] is employed as the input TBG shown in Fig. 1. As in [17, 21], an MSG instance I is defined by five parameters, i.e., problem dimension d; number of local optima n; standard deviation for each local optimum $\boldsymbol{\sigma}$ ($n \times 1$ vector); squeeze rate on each dimension for each local optimum $\boldsymbol{Q}$ ($n \times d$ matrix); and the ratio vector between each local optimum and the global optimum $\boldsymbol{r}$ ($n \times 1$ vector). Two parameters are set to fixed values: $d = d_0$ and $n = n_0$, and thus I =MSG$^{(d_0,n_0)}(\boldsymbol{\sigma}, \boldsymbol{Q}, \boldsymbol{r})$. Let $\boldsymbol{x} = \{\boldsymbol{\sigma}, \boldsymbol{Q}, \boldsymbol{r}\}$. Then, a problem instance is denoted by

$$I = MSG^{(d_0,n_0)}(\boldsymbol{x}) \tag{1}$$

DE [20] is employed as the EAPI, with parameters following the suggestion in [22]. The mission of DE is to find an $\boldsymbol{x}^*$, such that I^* is the best solution (the best required problem instance), where

$$I^* = \text{HFEBG-H}_{\text{DE}}(\text{MSG}^{(d_0,n_0)}(\boldsymbol{x}^*)) \tag{2}$$

Then, the optimized parameters $\boldsymbol{x}^* = \{\boldsymbol{\sigma}^*, \boldsymbol{Q}^*, \boldsymbol{r}^*\}$ can be used to re-construct the required problem instance.

Table 2: The settings of problem dimension d_0 and number of local optima n_0 in the three experiments.

		d_0	n_0
UE	$Exp\#1$	30	2
	$Exp\#2$	20	10
UD	$Exp\#3$	30	5

Three established algorithms with different evolving principles are employed, namely artificial bee colony (ABC) [23], composite differential evolution (CoDE) [24], and NBIPOP-aCMAES (CMA) [25]. Three experiments are implemented to generate both UE and UD problems for all three EAs, denoted by $Exp\#1$ to $Exp\#3$, as shown in Table 2. The maximum number of evaluations for each EA on a problem instance is set to $d_0 \times n_0 \times 500$. The number of independent runs is $R = 30$. The search range of generated problems is defined to be $[-1, 1]^{d_0}$. The total number of generated instances (i.e., the maximum evaluations of DE) is set to 1×10^3.

3.2 Experimental Results

A comparison between the U-test based measure of problem difficulty and the single-run based measure is performed in [21]. Experimental results reveal that the single-run based method may give inconsistent performance measure, due to the lack of statistical guarantee. It is common that an EA performs well on a problem in a single run but performs badly on the same problem in another single run. Statistical tests (e.g., U-test, H-test) filter out noise due to randomness on performance measure. An EA is applied to solve the problem repeatedly several times, and then the algorithm performance or problem difficulty is measured statistically. In the following, we investigate HFEBG-H on generating both UE and UD problems for each of the three algorithms employed.

3.2.1 Generating UE Problems

Table 3 shows the experimental results of $Exp\#1$. Three problems, namely UE-ABC, UE-CoDE and UE-CMA, are generated by HFEBG-H. The $fit.h_1$ column shows the p-value of multiple comparison between the target EA and the best performing EA of the other EAs. The data in the columns below the algorithm names represent the mean results the algorithm obtained in solving the problem. For example, for the problem UE-ABC, ABC obtains a mean result of 1.1768E-15 (averaged from 30 independent runs), while CoDE obtains 7.2803E-8 and CMA obtains 8.7718E-2. Apparently, ABC performs the best on this problem. In addition, ABC is significantly better than both CoDE and CMA, and the H-test p-value between ABC and CoDE is 1.2126E-6.

Suppose the confidence level is set as $\alpha = 0.01$. Then, 1.2126E-6 < 0.01 declares a significant difference, and thus the resulting problem UE-ABC is statistically uniquely easy for ABC.

Table 3: The resulting UE instances when $d_0 = 30$, $n_0 = 2$ (results for $Exp\#1$).

	$fit.h_1$	ABC	CoDE	CMA
UE-ABC	1.2126E-6	**1.1768E-15**	7.2803E-8	8.7718E-2
UE-CoDE	3.4436E-6	5.2564E-1	**1.3698E-6**	5.4456E-1
UE-CMA	5.2649E-5	5.6668E-5	2.7705E-7	**2.1039E-15**

Table 4: Rank of the results in Table 3 for easy comparison.

	ABC	CoDE	CMA
UE-ABC	**1**	2	3
UE-CoDE	2	**1**	3
UE-CMA	3	2	**1**

Table 5: The resulting UE instances when $d_0 = 20$, $n_0 = 10$ (results for $Exp\#2$).

	$fit.h_1$	ABC	CoDE	CMA
UE-ABC	2.4868E-5	**1.4439E-1**	2.6983E-1	2.9747E-1
UE-CoDE	7.5791E-5	4.6965E-2	**4.3937E-3**	1.0975E-1
UE-CMA	8.6999E-5	4.4043E-2	4.5498E-12	**2.9365E-15**

Note that the H-test p-value is calculated within the context of multiple comparison of all the three EAs, rather than a neck-to-neck comparison between two EAs. Table 4 gives the rank for easy comparison. Note that the ranks of non-target EAs may change when tested in another time, but the rank for the target EA is consistent. For example, for problem UE-ABC, if {ABC, CoDE, CMA} are employed to solve it again, ABC will consistently win the first rank, while for CoDE and CMA, their performance has no statistical guarantee, and thus either CoDE or CMA may rank the second. Table 5 shows the results of $Exp\#2$. It confirms that the UE problems generated in Table 3 can be repeated when the parameters d_0 and n_0 are changed.

3.2.2 Generating UD Problems

UD problems can be generated similarly ($Exp\#3$). Tables 6 and 7 show the resulting UD problems for algorithms and the rank, respectively.

Table 6: The resulting UD instances when $d_0 = 30$, $n_0 = 5$ (results for $Exp\#3$).

	$fit.h_1$	ABC	CoDE	CMA
UD-ABC	8.2873E-4	**4.7370E-15**	0	0
UD-CoDE	6.3517E-4	1.7919E-4	**3.2505E-1**	3.6188E-2
UD-CMA	1.6568E-4	5.2633E-4	8.7439E-7	**1.3463E-1**

Remaining a minimization problem within each component, the hierarchical fitness for generating UD problem is slightly differently assigned. In this case, the searching objective is to find a problem that only A_T finds difficult, but all other algorithms find easy.

1) If the mean result obtained by A_T is worse than all algorithms in $\boldsymbol{B_A}$, then $fit.h_1 = \max\{p_1, p_2, \ ... \ , p_{N-1}\}$, where $p_1 = H(A_T, B_1)$, $p_2 = H(A_T, B_2)$, ... , $p_{N-1} = H(A_T, B_{N-1})$. $fit.h_1$ represents the minimum performance difference between A_T and the worst performing EA in $\boldsymbol{B_A}$, while $fit.h_2$ and $fit.h_3$ are assigned a value of empty (Φ).

2) If the mean result obtained by A_T is worse than several EAs in $\boldsymbol{B_A}$, then $fit.h_2$ is assigned the negative number of EAs that beat A_T. The other two components are assigned the value of empty (Φ).

3) If A_T is better than all algorithms in $\boldsymbol{B_A}$, then $fit.h_3$ is assigned the residual value between the mean result obtained by A_T and the best performing EA in $\boldsymbol{B_A}$. Recall that the aim here is to generate a UD problem for A_T, but unfortunately A_T performs the best, thus $fit.h_3$ represents how much A_T lead ahead of other EAs. The other two components of fit are assigned the value of empty (Φ).

Except for the assignment of fit, the generation of UD problems is exactly the same as generation of UE ones, following the framework shown in Fig. 1.

Table 7: Rank of the results in Table 6.

	ABC	CoDE	CMA
UD-ABC	**3**	1.5	1.5
UD-CoDE	1	**3**	2
UD-CMA	2	1	**3**

4 Conclusions

In this paper, a systematic method for constructing performance comparison-based benchmark problems is proposed, namely the hierarchical-fitness-based evolving benchmark generator with the Kruskal-Wallis H-test (HFEBG-H). Performance of evolutionary algorithms is compared in terms of unique difficulty (specifically, uniquely easy (UE) and uniquely difficult (UD) problems). The UE and UD problems (to one algorithm) are obtained using a (meta-) evolutionary algorithm to maximize the performance difference between the algorithm and the other algorithms; meanwhile, the statistical guarantee is ensured by the Kruskal-Wallis H-test. A set of algorithms are employed. For each algorithm, UE/UD problems are evolved. A hierarchical fitness assignment method with H-test is designed to measure the unique difficulty of each problem instance. Experimental results verify that both UE and UD problems can be generated by the proposed method. The source code of the proposed work is available in [26].

Acknowledgement

The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 125313]. We thank Prof. Hideyuki Takagi for the meaningful discussions and suggestions on the statistical issue.

References

[1] J. Rönkkönen, X. Li, V. Kyrki, and J. Lampinen, A framework for generating tunable test functions for multimodal optimization, *Soft Comput.*, 15(9): 1689–1706, 2011. Doi:10.1007/s00500-010-0611-1.

[2] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis, Towards objective measures of algorithm performance across instance space, *Comput. Oper. Res.*, 45:12–24, 2014. Doi:10.1016/j.cor.2013.11.015.

[3] K. Smith-Miles, Kate and L. Lopes, Measuring instance difficulty for combinatorial optimization problems, *Comput. Oper. Res.*, 39(5):875–889, 2012. Doi:10.1016/j.cor.2011.07.006.

[4] B.Y. Qu, J.J. Liang, Z.Y. Wang, Q. Chen, and P.N. Suganthan, Novel benchmark functions for continuous multimodal optimization with comparative results, *Swarm Evol. Comput.*, 26:23–34, 2016. Doi:10.1016/j.swevo.2015.07.003.

[5] Black-Box-Optimization-Benchmarking (BBOB), 2009 [online] http://coco.gforge.inria.fr/doku.php (Accessed 2016)

[6] X. Yao, Y. Liu, and G. Lin, Evolutionary programming made faster, *IEEE Trans. Evol. Comput.*, 3(2):82–102, 1999. Doi:10.1109/4235.771163.

[7] M. Gallagher and B. Yuan, A general-purpose tunable landscape generator, *IEEE Trans. Evol. Comput.*, 10(5):590–603, 2006. Doi:10.1109/TEVC.2005.863628.

[8] S. Das and P.N. Suganthan, Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems, [technical report] Jadavpur University, Kolkata 700 032, India; Nanyang Technological University, Singapore 639798, Singapore, 1–42, 2011.

[9] J.H. Moore, M. Shestov, P. Schmitt, and R.S. Olson, A heuristic method for simulating open-data of arbitrary complexity that can be used to compare and evaluate machine learning methods, *Proceedings of the Pacific Symposium*, pp.259–267, 2018.

[10] D.H. Wolpert and W.G. Macready, No free lunch theorems for optimization, *IEEE Trans. Evol. Comput.*, 1(1):67–82, 1997. Doi:10.1109/4235.585893.

[11] J.N. Hooker, Testing heuristics: We have it all wrong, *J. Heuristics.*, 1(1):33–42, 1995. Doi:10.1007/BF02430364.

[12] J. Branke and C.W. Pickardt, Evolutionary search for difficult problem instances to support the design of job shop dispatching rules, *Eur. J. Oper. Res.*, 212(1):22–32, 2011. Doi:10.1016/j.ejor.2011.01.044.

[13] J.I. van Hemert, Evolving combinatorial problem instances that are difficult to solve, *Evol. Comput.*, 14(4):433–462, 2006. Doi:10.1162/evco.2006.14.4.433.

[14] D.S, Himmelstein, C.S. Greene, and J.H. Moore, Evolving hard problems: Generating human genetics datasets with a complex etiology, *BioData Min.*, 4(1):1–13, 2011. Doi:10.1186/1756-0381-4-21.

[15] W.B.F. Langdon and R. Poli, Evolving problems to learn about particle swarm optimisers and other search algorithms, *IEEE Trans. Evol. Comput.*, 11(5): 561–578, 2007. Doi:10.1109/TEVC.2006.886448.

[16] S. Shirakawa, N. Yata, and T. Nagao, Evolving search spaces to emphasize the performance difference of real-coded crossovers using genetic programming, *IEEE Congr. Evol. Comput.* (*CEC*) 2010. Doi:10.1109/CEC.2010.5586065.

[17] Y. Lou, Techniques for improving online and offline history-assisted evolutionary algorithms, PhD Thesis, *City University of Hong Kong*, Hong Kong, 2017.

[18] M.P. Fay and M.A. Proschan, Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules, *Stat. Surv.*, 4:1–39, 2010. Doi:10.1214/09-SS051.

[19] W.H. Kruskal and W.A. Wallis, Use of ranks in one - criterion variance analysis, *J. Am. Stat. Assoc.*, 47(260): 583–621, 1952.

[20] R. Storn and K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, *J. Glob. Optim.*, 11(4): 341–359, 1997. Doi:10.1023/A:100820282.

[21] Y. Lou and S.Y. Yuen, On constructing alternative benchmark suite for evolutionary algorithms, *Swarm Evol. Comput.* (In Press) 2018. Doi:10.1016/j.swevo.2018.04.005.

[22] R. Storn, [online] Differential evolution homepage. http://www1.icsi.berkeley.edu/∼storn/code.html. (Accessed 2017)

[23] D. Karaboga, Dervis and B. Basturk, A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm, *J. Glob. Optim.*, 39(3): 459–471, 2007. Doi:10.1007/s10898-007-9149-x.

[24] Y. Wang, Z. Cai, and Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, *IEEE Trans. Evol. Comput.*, 15(1): 55–66, 2011. Doi:10.1109/TEVC.2010.2087271.

[25] I Loshchilov, CMA-ES with restarts for solving CEC 2013 benchmark problems, *IEEE Congr. Evol. Comput.* (*CEC*) 2013, pp.369–376, 2013. Doi:10.1109/CEC.2013.6557593.

[26] S.Y. Yuen, [online] Source code. http://www.ee.cityu.edu.hk/∼syyuen/Public/Code.html. (Accessed 2018)