Investigating Benchmark Correlations when Comparing Algorithms with Parameter Tuning (Detailed Experiments and Results)

Lee A. Christie
Division of Computing Science and Mathematics
University of Stirling
Stirling, UK
lee.christie@stir.ac.uk

Alexander E.I. Brownlee
Division of Computing Science and Mathematics
University of Stirling
Stirling, UK
alexander.brownlee@stir.ac.uk

John R. Woodward
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK
j.woodward@qmul.ac.uk

ABSTRACT

Benchmarks are important to demonstrate the utility of optimisation algorithms, but there is controversy about the practice of benchmarking; we could select instances that present our algorithm favourably, and dismiss those on which our algorithm underperforms. Several papers highlight the pitfalls concerned with benchmarking, some of which concern the context of the automated design of algorithms, where we use a set of problem instances (benchmarks) to train our algorithm. As with machine learning, if the training set does not reflect the test set, the algorithm will not generalize. This raises some open questions concerning the use of test instances to automatically design algorithms.

We use differential evolution, and sweep the parameter settings to investigate the practice of benchmarking using the BBOB benchmarks. We make three key findings. Firstly, several benchmark functions are highly correlated. This may lead to the false conclusion that an algorithm performs well in general, when it performs poorly on a few key instances, possibly introducing unwanted bias to a resulting automatically designed algorithm. Secondly, the number of evaluations can have a large effect on the conclusion. Finally, a systematic sweep of the parameters shows how performance varies with time across the space of algorithm configurations.

CCS CONCEPTS

•Computing methodologies → Search methodologies;

KEYWORDS

benchmarks, BBOB, ranking, differential evolution, continuous optimisation, parameter tuning, automated design of algorithms.

This technical report is provided in support of the paper "Investigating Benchmark Correlations when Comparing Algorithms with Parameter Tuning" published at GECCO 2018, Kyoto, Japan.

1 INTRODUCTION

Continuous optimisation is the practice of sampling a continuous search space, with the aim of minimizing (or maximising) an objective. Continuous optimisation benchmark sets are typically used to compare the performance of different metaheuristics. The Black-Box Optimization Benchmarking (BBOB-2009) benchmarks [11] are a commonly used benchmark for comparing continuous optimisation metaheuristics [18]. BBOB consists of 24 noiseless test functions and 30 noisy test functions. While it is well known that no single algorithm will perform well across all functions we ask *how good are the benchmarks at teasing out the different performances of different algorithms?* Furthermore, when we consider the automatic design of algorithms (ADA), the set of functions used to train the algorithms is critical. As with machine learning, if the training set does not reflect the test set, then the algorithm will not generalize. Consequently, it is very important to understand how algorithm performance varies across the benchmarks before extending our focus to real-world problems.

We question whether correlations in performance of different algorithms can identify benchmarks that are similar in terms of performance. Therefore, correlated benchmarks will not contribute any new information to the overall picture of performance of an algorithm. We consider algorithms with different parameter setting to be different algorithms, and therefore, the algorithms defined by the set of parameters defines the algorithm design space.

In this paper we make three major contributions:

(1) Firstly, algorithm performances on several benchmark functions are highly correlated (within the set of algorithms we consider). This may lead to the false conclusion that an algorithm performs well in general when it only performs poorly on a few key instances.
(2) Secondly, the number of evaluations has a dramatic impact when concluding either which algorithm performs best, or which benchmarks are correlated and therefore which instances to select when automatically designing an algorithm.
(3) Finally, we conducted a two-stage (coarse-grained then fine-grained) systematic sweep of the parameters to investigate how performance varies with number of evaluations for different parameter settings.

We begin the paper in Section 2 with a summary of the background in benchmarking and automatic design of algorithms, and their common issues. We make some formal definitions and explain our experimental design in Sections 3 and 4. In Section 5 we report experimental results, with a discussion of their implications in Section 6. Finally, in Section 7 we draw our conclusions and suggest future work arising from this study.

2 BACKGROUND

2.1 Benchmarking

Benchmarking is essential to demonstrate the performance of the algorithm on a set of problems. In this paper, we are considering ADA, where we need two sets of instances; one to tune the algorithm parameters, and one to test the algorithm.

In machine learning, the central assumption is that the training data is drawn from the same distribution as the test data. We cannot expect to build a good quality model from poor quality data. Therefore, we could make the statement "an algorithm is only as good as the benchmarks it has been tested on". In general, testing on more benchmarks is better than testing on fewer benchmarks, and gives us more confidence that the algorithm is effective. It has also been shown recently that the typical approach of sampling uniformly at random to generate benchmark instances does not in fact yield a uniform distribution of instances, leading to further possible bias in the results [7]. While from a machine learning perspective, we gain more confidence in a model as it is demonstrated to be accurate on more data, we can never have total confidence in its performance on unseen data. The aim of this paper is to show that selecting test cases is problematic in general, and so we should be aware of these issues when automatically designing algorithms.

2.2 Automated Design of Algorithms

ADA includes parameter tuning, automated algorithm selection, automated algorithm configuration, and synthesising algorithmic components from primitive instructions. In one sense, ADA means the tuning of numerical parameters to obtain good performance on a future set of problem instances. Examples are mutation rate and crossover rate in genetic algorithms. ADA can also mean algorithm configuration, where a choice of algorithmic components needs to be made. Examples are deciding on a steady-state or generational population model in evolutionary algorithms. Another example is elitism, where the best individuals in the current population are promoted to the next generation without undergoing selection. Another interpretation of ADA is a GP as a hyper-heuristic [30], where algorithmic components of an existing algorithm is created from scratch by GP. An example is designing job shop scheduling rules [6] and evolving heuristics for examination timetabling problem [22].

More recently, there has been extensive interest in automatically tuning or selecting algorithms by modelling algorithm performance with respect to the fitness landscape or other features of the problems. For example, [17, 21] showed that only a small number of features are needed to separate the BBOB problem groups. These can then be used to select well-suited algorithms [2, 5], predict performance [4], or choose parameter settings [3, 19]. More relevant to the present work, [26, 27] showed how parameter tuning for Differential Evolution (DE) for the BBOB and CEC2014 functions was impacted by different running budgets. The correlations we find may provide some further explanation for this.

While all of these approaches can be considered to be ADA, we will specifically look at two parameters of differential evolution [25]; differential weight (F), and crossover rate (CR). An interesting new direction is an investigation into the relationship between reasonable parameter settings on a set of benchmarks which are described using a set of features. A model can then be built to compute the parameter setting for a new function. Whichever of these methods we employ to automatically design algorithms, in each of the cases choosing the appropriate problem instances is crucial, as these functions form the basis on which we make any empirical claims.

2.3 Benchmarking, ADA and No Free Lunch

The no free lunch theorems [29] state that no algorithm performs better than any other algorithm over the set of all discrete problems. This is also demonstrated empirically by the work of [20] where the performance of algorithms is mapped into the feature space of problems. This research, very visually, shows there are clusters of expertise where algorithms outperform each other, and highlights the fact that certain algorithms are specialised to certain sets of problems. This work also demonstrates the much broader interpretation of the no free lunch theorems: on a subset of problems there is an algorithm with dominant performance. ADA aims to find this algorithm with dominant performance. To reiterate our main point, picking a set of problem instances is very important in the context of algorithm design in general, but ADA in particular. In general, we are not interested in solving the set of all problems, but solving problems with a particular characteristic in common. These characteristics are represented by a sample of the benchmarks instances. As [8] reports, performance results of an algorithm depend as much on the benchmarks as they do on the algorithm. Essentially, NFL is a bridge between algorithms and benchmarks.

2.4 Benchmarking is Problematic

There are a lot of issues surrounding benchmarking, and [12] lists principles regarding benchmarking. Of particular interest is benchmarking with GP [28] as GP is one ADA technique. One of the dangers of benchmarking is not investing the same amount of evaluations in tuning two algorithms; you may invest considerable time in tuning one algorithm, while you invest little or no time to the second algorithm, or just use the default settings. Tools like *irace* [15], reduce this possibility by allowing both algorithms the same amount of "warm-up" computational time tune before being applied to the final set of test benchmark instances, when the true (and fair) comparison can be made.

A further issue is that typically comparisons take the form of "Algorithm A outperformed B on 20/24 of the benchmarks, while B outperformed A on the remaining 4". If those 20 benchmarks are highly correlated, yet the other 4 are not, then B can be said to be more general than A. Of course, the usual conclusion of such a paper is that A was most general, performing better on more instances. Our paper is concerned with investigating the tuning of parameters on a set of benchmarks and examining the correlations in performance between the algorithms on different functions.

3 DEFINITIONS

We define an algorithm $A = (M, \theta)$ as a metaheuristic M with a parameter set θ if applicable. The same metaheuristic with a different parameter set θ' (or different operators) is regarded as a different algorithm A'. The algorithms we consider are stochastic, and so will produce different results from different runs. This is

dependent on the pseudo-random number generator (PRNG) seed s. We denote A_s to be algorithm A run with PRNG seed s. The seed s may be set to a specific value for repeatability, or set based on system time.

We define a single benchmark $B = (f, \Omega, \phi, L)$ as a function f, search space Ω, parameter set ϕ if applicable, an evaluation limit L. For a given function, the minimum fitness $\min_x (f(x))$ may be known or unknown. If we are using an evaluation limit L, we do not need to know the minimum fitness as termination is controlled by the evaluation limit and not controlled by convergence within a tolerance of the global optimum. For some benchmarks such as the BBOB benchmark set, the space is rotated and/or translated, and so is also dependent on an PRNG seed t. We denote B_t to be benchmark B instance constructed with PRNG seed t.

The fitness of a single run we denote as $A_s(B_t)$ to be the fitness reached when seeded algorithm A_s is run on seeded benchmark instance B_t. Since the PRNG seeds are fixed this will produce the same fitness value if the algorithm is re-run on the benchmark. It is conventional for the seed t be regenerated for each run to allow for a statistical analysis of the algorithm performance to be undertaken. We conform to this convention.

Typically, within the literature [10], the expected number of function evaluations to convergence within a defined tolerance on the global optimum is used as a measure of algorithm performance. This is sometimes called the *fixed-target* performance measure [9].

As we are optimising the parameters of an algorithm and measuring the performance of the algorithm on the problem (i.e. the algorithm's fitness for the problem), to avoid confusing with *fitness* of a candidate solution, we refer to measuring the performance of an algorithm as meta-fitness. We define *meta-fitness* to be how well algorithm A performs on benchmark B according to *some* measure M. In our experiments, M is the expected quality of the best candidate found after a predefined number of fitness evaluations has been reached (*fixed-budget*).

The advantage of using a fixed-budget for M, over the more conventional fixed-target, is that experiments generally run for less time, as convergence is not required. Hence this method is often used in parameter tuning (e.g. methods such at iterated racing [15]).

However, there are drawbacks using a fixed-budget. Fixed-target measure allows for ratio data, meaning the numerical values of the runtime, to be compared [9]. For instance, it may be said that on benchmark B, algorithm A performs *twice as fast* as algorithm A'. Fixed-budget uses ordinal data so ratios are meaningless. For instance, it may be said that on benchmark B, algorithm A outperforms algorithm A', but not by how much. We take this drawback in to account in the interpretation of results.

Sampling $A_s(B_t)$ for random (s, t) gives rise to a probability distribution. We denote $A(B)$ as the mean of this distribution, which we call the *meta-fitness* of A on B. Since this is not known, we denote $\overline{A(B)}$ as an estimate of $A(B)$ which is computed by averaging over n runs using the usual arithmetic mean as $\overline{A(B)} = \frac{\sum A_s(B_t)}{n}$ for n choices of (s, t). We are taking $\overline{A(B)}$ to be a good approximation to $A(B)$ for large n, as is standard practice. This average is used as the meta-fitness (the expected measure of algorithm performance on a given benchmark).

Table 1: Differential Evolution parameters held constant in all experiments.

Parameter	Setting
Population Size	10
Convergence Tolerance	0.0
Differential Weight Dithering	Off
Polish (L-BFGS-B)	Off
Population Initialization	Latin Hypercube sampling

Table 2: Differential Evolution parameters ranges varied for coarse-grained parameter sweep.

Parameter	Range of Settings
differential weight (F)	$\{0.0, 0.1, \ldots, 1.9\}$
crossover probability (CR)	$\{0.0, 0.1, \ldots, 0.9\}$

4 EXPERIMENTAL DESIGN

The focus of our study is a set of different parameters (differential weight, F and crossover probability, CR) for differential evolution, applied to the BBOB benchmarks. The experiment was in two stages to keep the total run time to a practical level. Stage 1 was a coarse-grained sweep of the parameters, covering the full range of possible values, with the aim of finding a suitable near-optimal range for each to study in more detail. Stage 2 was a fine-grained sweep of the parameters surrounding the previously-identified near-optimal setting, the results from which form the major basis for our discussion.

4.1 Algorithm and Benchmark Set-up

We used the DE implementation in the SciPy[1] optimize package. For all experiments, the DE parameters in Table 1 were held constant.

The benchmark functions used are the Python implementation[2] of the BBOB functions, converted to Python 3 using the `2to3` tool[3]. The BBOB benchmarks used were the 24 noiseless functions, on the 10-dimensional search space $[-5, 5]^{10}$.

4.2 Coarse-Grained Parameter Sweep

For coarse-grained sweep, a generation limit of 10 generations was applied. In this implementation of DE, this constraint was set by setting `maxiter` to 9, which specified how many times the algorithm will iterate after generating a population. For each parameter setting θ, the performance $\overline{A(B)}$ was measured, on termination, for $n = 1000$ to obtain the estimate $\overline{A(B)}$.

4.3 Choosing Optimal Parameters

For each algorithm configuration θ, each of the 24 benchmarks provide a ranking of which configurations were best, by giving an ordering of the configurations from best to worst. To reach a consensus of the best configuration over all benchmarks, the rankings were used a ballots in an instant run-off or preferential voting system [1]. In this system, the counting is conducted in rounds.

[1] SciPy 0.18.1 for Python 3
[2] https://github.com/numbbo/coco
[3] https://docs.python.org/3/library/2to3.html

At each round, the configuration with the lowest number of first-ranks is eliminated and the algorithms are re-ranked (effectively the benchmarks 'voting' for that algorithm as their first preference then transfer their vote to the next preference). The algorithm remaining once all others are eliminated is chosen.

4.4 Local Space of Parameter Values

At the fine-grained stage, we chose a neighbourhood region of 10×10 samples centred around the best parameter setting as determined by the voting following the coarse grained parameter sweep. The neighbourhood samples were spread out in increments of 0.01 for each function. For this experiment, the DE was run for 25 generations (by setting `maxiter` to 24). In contrast to the course-grained parameter sweep (which only recorded best fitness on termination) we recorded the best fitness reached at each generation. The mean for each configuration at each generation was computed over the 1000 samples.

5 RESULTS

The data for these experiments is available from [TBC] [4]. With the initial coarse-grained parameter sweep, the rankings of parameter settings produces F = 0.3, CR = 0.9 as overall consensus optimum parameters. This was done by instant run-off voting as described, however, the same consensus is also obtained by averaging the ranks. The resulting 10×10 region for the local parameter sweep was $F \in \{0.25, 0.26, \ldots 0.34\}$ $CR \in \{0.85, 0.86, \ldots 0.94\}$ as shown in Fig. 1.

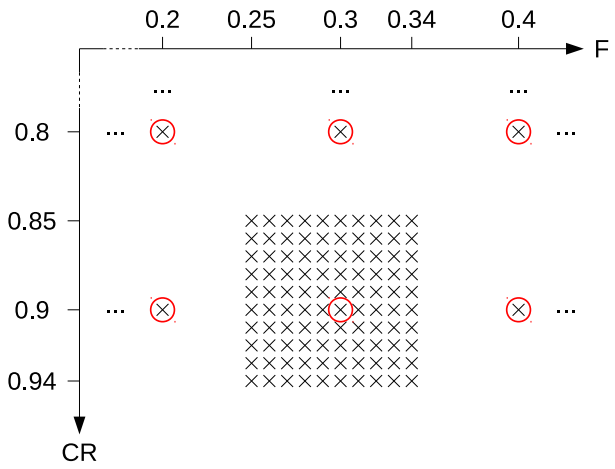

Figure 1: View of parameter space for differential weight F and crossover rate CR in the fine-grained parameter sweep. The red circles indicate the parameters run in the coarse-grained parameter sweep and the black crosses indicate the parameters run in the fine-grained parameter sweep.

We have 100 DE parameter configurations for each function. A standard way of presenting the result would be to plot the $\overline{A(B)}$ against L. For benchmark F1, we have chosen three parameter configurations and plotted the meta-fitness against generation in Fig. 2.

We wish to compare the performance of different algorithms across different functions. So next we introduce the correlations for different DE configurations. For a given generation limit L, each of DE configuration θ has a certain estimate of meta-fitness $\overline{A(B)}$ for each benchmark B. We consider the functions pairwise. For a pair of benchmarks B and B' which differ only by parameter setting ϕ, we can map θ to a point (x, y), where $x = \overline{A(B)}$, $y = \overline{A(B')}$. The value $\rho(L)$ is the Spearman's rank correlation between the two data vectors $\vec{x}$ (the vector of x coordinates) and $\vec{y}$ (the vector of y coordinates), which is a function of generation limit L. We are using Spearman's rank correlation rather than the more common Pearson correlation.

This is because we are comparing meta-fitness between different functions, where we cannot assign meaning to the numerical values, and only wish to compare the performance on an ordinal basis. In Fig. 3 we have plotted ρ for the first few generations for functions F1, and F6. We see that the correlation starts near zero at generation 0, as the first generation is random, then rises and falls with generation limit.

Since we have 24 functions to compare pairwise, we can construct a correlation matrix between the functions. Here, each cell corresponds to the correlation in meta-fitness between the two functions in the row and column header. We also calculate the median value of correlation between a given benchmark B with every other benchmark. The correlation matrix for generation limit 10 is shown in Table 3.

Since the median correlation for each benchmark is a function of generation limit L, we now plot median Spearman's rank correlation of each of the 24 functions against generation limit L as in Fig. 4.

In Fig. 4 we observe that in generation 2, *all* functions are strongly positively correlated with one-another, except for function F23 which has a median correlation of -0.30.

The lowest median correlation observed is function F7 at generation 5, where median correlation is -0.90. A number of other functions: F17, F18, and F24 are also low at this generation, with other functions F6 and F19 dropping to near-zero correlation.

At generation 8, a crossing point has occurred where many functions with high median correlations such as F1, F12, F13, and F16 now drop to negative correlations, and many functions with previously negative median correlation such as F7, F17, and F18 now move to having high positive median correlations. We see that functions F7, F17, and F18 maintains a strong positive correlation with one another as shown in Fig. 5, where as generation limit increases, the three approach a correlation near 1.0 with one another. Thus, these three functions (within the scope of which we have studied them) give us the same information about the algorithm's performance, and so it is redundant to run all three in this case.

For most generation limits L during these runs, most functions are on average positively correlated with a few exceptions.

[4] Link to our Online Research Repository will be added here.

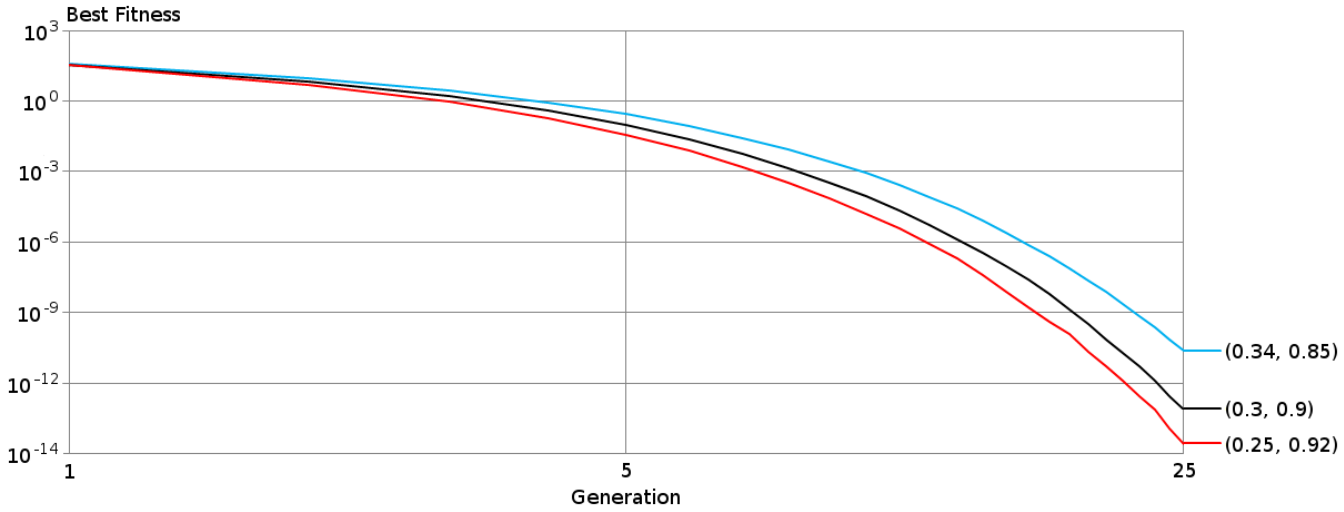

Figure 2: Fitness by generation (log-log plot) for three selected DE parameters settings on function F1. The three parameters are the suite consensus best $\theta = (\mathbf{F} = 0.3, \mathbf{CR} = 0.9)$, and the best $(0.25, 0.92)$ and worst $(0.34, 0.85)$ within the fine-grained 10×10 parameter sweep. Fitness is the mean over 1000 runs.

Table 3: Matrix of correlations between all benchmark functions at generation 10. Blue cells denote a strongly-positive correlation, red cells denote a strongly-negative correlation. The right-most column is the median correlation of the related function with every other function.

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24		Median
	0.98	-0.76	-0.84	-0.81	-0.93	-0.96	-0.37	0.19	-0.94	-0.89	-0.28	0.55	0.94	-0.81	0.95	-0.92	-0.94	-0.25	-0.5	-0.44	-0.58	-0.52	0.94	F1	-0.52
		-0.75	-0.81	-0.85	-0.9	-0.92	-0.39	0.27	-0.93	-0.88	-0.26	0.52	0.94	-0.78	0.93	-0.88	-0.94	-0.31	-0.49	-0.35	-0.48	-0.54	0.94	F2	-0.49
			0.92	0.71	0.75	0.83	0.04	-0.27	0.79	0.81	0.09	-0.77	-0.78	0.98	-0.89	0.88	0.87	0.16	0.49	0.49	0.39	0.33	-0.9	F3	0.39
				0.78	0.92	0.9	0.12	-0.14	0.84	0.9	0.02	-0.79	-0.88	0.96	-0.95	0.95	0.94	0.21	0.3	0.53	0.39	0.47	-0.93	F4	0.39
					0.79	0.75	0.27	-0.1	0.85	0.88	-0.04	-0.71	-0.77	0.76	-0.84	0.83	0.85	0.36	0.47	0.5	0.36	0.59	-0.88	F5	0.47
						0.94	0.25	-0.09	0.84	0.92	0.14	-0.6	-0.96	0.84	-0.95	0.93	0.96	0.35	0.31	0.42	0.35	0.5	-0.92	F6	0.35
							0.18	-0.07	0.9	0.88	0.35	-0.62	-0.95	0.88	-0.98	0.94	0.95	0.32	0.49	0.47	0.5	0.35	-0.95	F7	0.47
								-0.54	0.24	0.27	0.03	0.12	-0.27	0.09	-0.2	0.16	0.25	-0.04	0.04	0.31	0.26	0.79	-0.18	F8	0.12
									-0.12	-0.25	0.15	0.07	0.21	-0.22	0.14	-0.19	-0.21	0.39	-0.05	-0.01	-0.03	-0.5	0.15	F9	-0.07
										0.83	0.15	-0.62	-0.83	0.79	-0.92	0.88	0.88	0.15	0.47	0.38	0.7	0.49	-0.94	F10	0.47
											0.03	-0.79	-0.9	0.89	-0.94	0.98	0.95	0.22	0.48	0.62	0.35	0.56	-0.92	F11	0.48
												0.09	-0.32	0.08	-0.19	0.09	0.18	0.55	0.28	0.07	0.03	-0.43	-0.18	F12	0.07
													0.58	-0.81	0.71	-0.81	-0.68	-0.09	-0.33	-0.71	-0.26	-0.26	0.7	F13	-0.33
														-0.85	0.95	-0.92	-0.98	-0.42	-0.39	-0.39	-0.28	-0.42	0.93	F14	-0.42
															-0.94	0.94	0.93	0.24	0.47	0.58	0.33	0.39	-0.93	F15	0.39
																-0.98	-0.99	-0.31	-0.5	-0.52	-0.45	-0.45	0.99	F16	-0.50
																	0.96	0.19	0.49	0.59	0.44	0.47	-0.95	F17	0.47
																		0.36	0.45	0.48	0.35	0.49	-0.98	F18	0.45
																			0.19	0.14	-0.39	-0.19	-0.32	F19	0.16
																				0.44	0.35	0.04	-0.54	F20	0.30
																					0.27	0.38	-0.45	F21	0.38
																						0.38	-0.47	F22	0.33
																							-0.44	F23	0.35
																								F24	-0.47

6 DISCUSSION

6.1 Summary of Benchmark Study

In summary, we have taken a standard set of benchmarks for continuous optimisation used in competitions and research papers. We have run a standard implementation of DE, performing a sweep across the parameters, $\mathrm{F} \in \{0.0, 0.1 \ldots, 1.9\}$, and $\mathrm{CR} \in \{0.0, 0.1, \ldots, 0.0\}$, while fixing other parameters (see Table 2).

We ranked the performance of each parameter settings across the functions to determine reasonable parameter settings for these functions. Having identified the best setting as $\mathrm{F} = 0.3$, $\mathrm{CR} = 0.9$ for the benchmark suite, we then conducted a fine-grained investigation of the parameter settings around the best values where we swept $\mathrm{F} \in \{0.25, 0.26, \ldots, 0.34\}$, and $\mathrm{CR} \in \{0.85, 0.86, \ldots, 0.94\}$.

We found that early in the run there was very little correlation between the performances of different parameter settings. This is to be expected as the first generation of DE is effectively a random search. However, after a few generations, most functions are highly correlated, while some are anti-correlated. As the number of generations increases we see different performances across the functions. The performance on some functions remains correlated over the

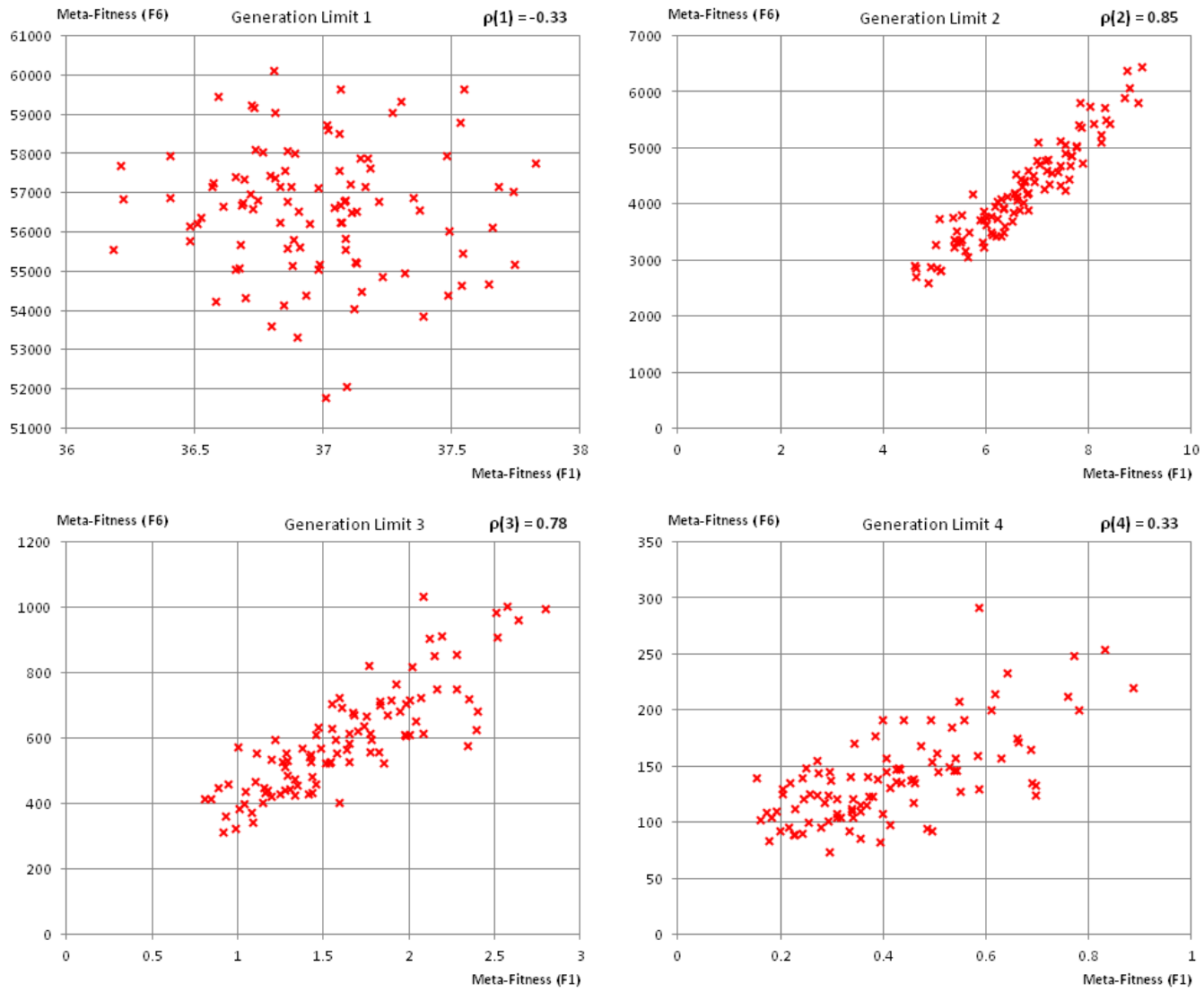

Figure 3: All 100 DE configurations plotted by meta-fitness on function F1 and meta-fitness on function F6, for generation limit of 1 (top-left, $\rho = -0.33$), 2 (top-right, $\rho = 0.85$), 3 (bottom-left, $\rho = 0.78$), and 4 (bottom-right, $\rho = 0.33$). Note that the scales on both axes decreases with generation, as the algorithms advance.

whole run, while other functions become anti-correlated, and then correlated again.

6.2 Implications for Automated Design of Algorithms

Benchmarking is necessary for at least two reasons in the context of ADA. Firstly, we need to provide a set of example benchmark instances (problems, functions or test data) on which we *train* our model using ADA. Secondly, we need to demonstrate the utility of our automatically designed model on a separate set of instances. This is in contrast to manual design, where we only need to demonstrate the performance of an algorithm on the set of *test* instances, as is done in most papers.

These two sets of instances (training and test) need to be related in some way if there is to be any meaningful connection between the trained model and its performance on the test set. In machine learning, this is well understood, where train and test performances are reported. This division of data into a training and test set is less well understood by in optimisation community, where we often have a small finite set of optimisation problems which are not explicitly divided into training and test set.

The performance of DE on a number of functions in the BBOB set are highly correlated (e.g. F7, F17, F18), and therefore could be considered as redundant. What is important about these three functions, for example, is that the performance of the algorithm is

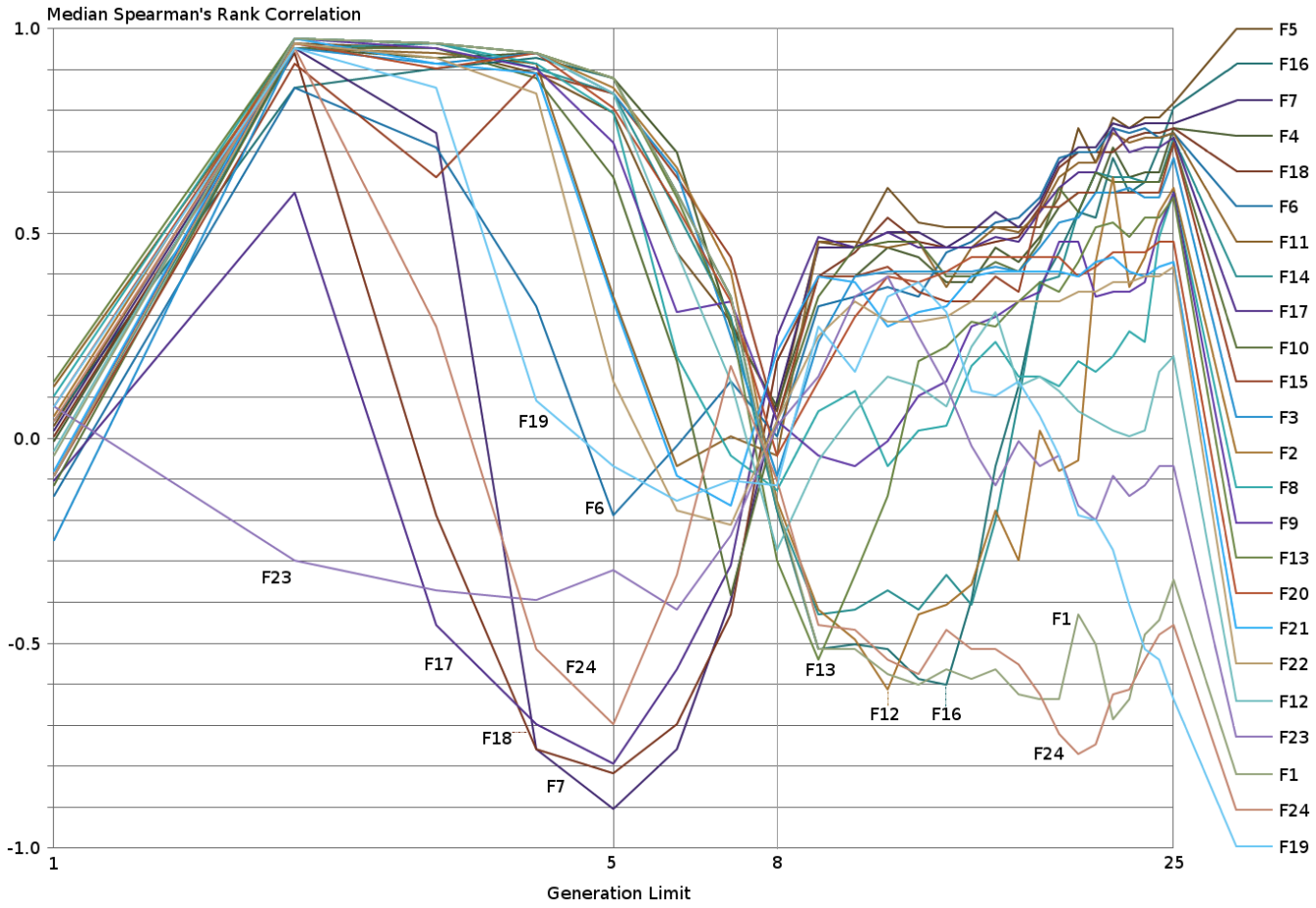

Figure 4: The median Spearman's rank correlation by generation (log-x) limit on all BBOB benchmarks of dimension 10, for DE using a parameter sweep around $\theta = (F = 0.3, CR = 0.9)$. Generation 8 has been highlighted as the crossing point.

highly correlated at each "snap shot" in the runs (of course excluding the initial few populations which are effectively random search and therefore not correlated).

What is more other functions appear to be strongly negatively correlated. These correlations do not necessarily just occur within the five groups of functions based on their structure (defined by [11]): corroborating earlier results [18] that clustered the BBOB functions in to different groups according to algorithm performance. However, in addition to this observation, we have also found that these correlations vary with the number of function evaluations. This has a particular implication for using exploratory landscape analysis to predict performance and select appropriate algorithms as advocated by [18]. Choice of the evaluation budget must match the available budget in the target "unseen" instances, or the performance model will be flawed, biased by correlations present only in part of the search space.

Therefore, we should take care when synthetically generating problem instances for ADA, and then make claims about the applicability of the algorithm to real-world problems. While the results in this paper concern one algorithm, DE, applied to one domain, continuous optimization, it is possible that these conclusions extend to other algorithms applied to other domains. While benchmarking is tricky, we should adopt a scientific approach and report results as fully as possible; specifically giving both the number of function evaluations and precise specifications of problem instances, at each stage of the experimental process. Furthermore, it is false to assume that parameter tuning on a smaller evaluation budget will lead to fair comparisons with a larger budget [26, 27].

7 CONCLUSION AND FUTURE WORK

We now discuss our results which naturally lead onto some related further work. The current work only examines DE, a well-known algorithm for continuous optimisation. However, to draw broader conclusions, we intend to investigate at how benchmark functions compare under other algorithms, such as CMA-ES and conventional genetic algorithms. We suspect similar conclusion can be drawn for algorithms other than DE (preliminary experiments with a GA show the same pattern), but what remains to be seen is if functions which are correlated when using DE are also correlated when using, for example, CMA-ES.

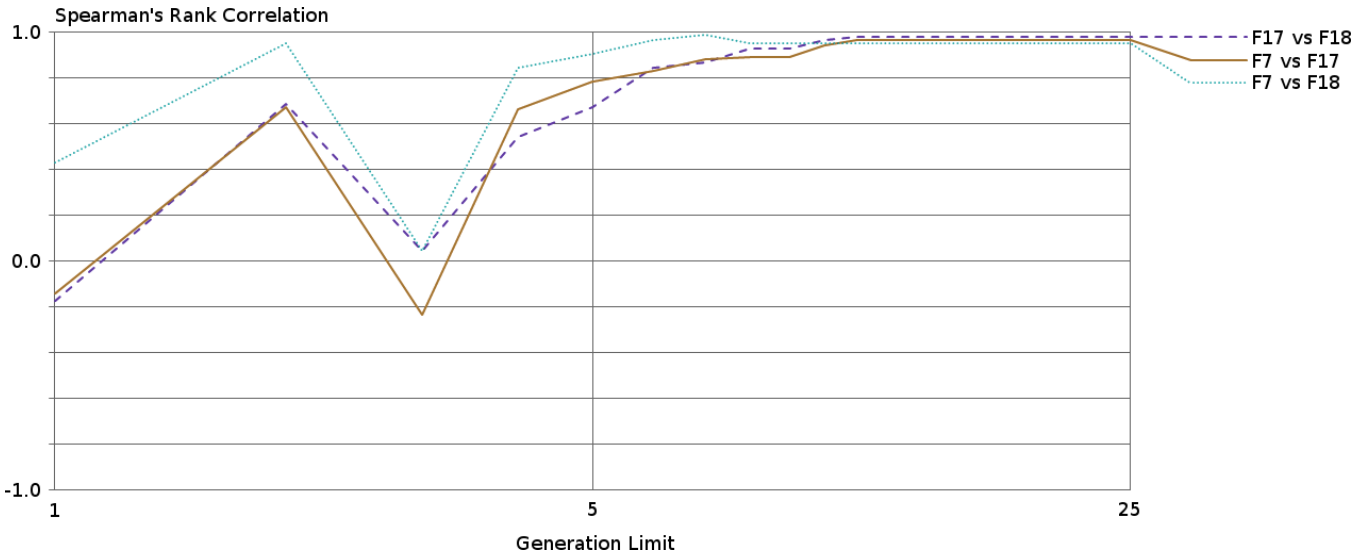

Figure 5: The pairwise correlations by generation (log-x) between BBOB benchmark functions F7, F17, and F18 of dimension 10, for DE using a parameter sweep around $\theta = (F = 0.3, CR = 0.9)$.

We have used a fixed-budget (i.e. a predefined number of function evaluations) to compare performance on different functions. We should also repeat our analysis using the approach of obtaining solutions within a certain tolerance of the global optima. It would be interesting to see if functions which are correlated using a fixed-budget are also correlated using a fixed-tolerance limit. Similarly, we aim to conduct a similar analysis holding different DE parameters fixed and allowing other parameters to become part of the ADA process.

In Section 5, we identified that for the scope of the study, functions F7, F17, and F18 remained very strongly positively correlated. Thus, given the performance of DE on one of these functions, the other two add no information to the benchmarking process This corresponds to the findings of [18] in which F17, and F18 were always clustered together. If we are able to identify highly-correlated benchmark functions in terms of algorithm performance, we may be able to identify a procedure to select and eliminate redundant functions from a benchmark set. A possible procedure for identifying a smaller set of representative benchmarks from a suite of benchmarks may involve clustering benchmarks based on correlations, then eliminating all but the centre of those clusters. This would produce a benchmark set which is as powerful as the original (in terms of describing algorithm performance), but are a smaller set of functions. In Section 2 we discussed that having benchmarks which are very similar (in terms of algorithm performance) dilutes the conclusions we can draw when an algorithm performs "well" on x out of 24 benchmarks. Complicating the picture further is that these correlations may be algorithm-specific.

If we are able to identify highly-correlated benchmarks, we may also be able to identify possible combinations of performance features which do not exist in the benchmark sets. These gaps could be filled by generating new benchmarks. Some work has already been done in this area. For example, it is possible to generate seed points corresponding with a desired underlying structure, and then interpolating a fitness landscape between them [13, 16]. It is also possible to evolve instances matching a particular distribution of features [24], or corresponding to a particular order of inter-variable dependencies [23]. [14] have evolved problems to reveal weaknesses in PSO algorithms. This would have the potential to form a basis for a general approach to constructing new benchmarks that would mitigate the problem of correlated performance on existing ones.

8 ACKNOWLEDGEMENT

Work funded by UK EPSRC [grants EP/N002849/1, EP/J017515/1]. Results obtained using the EPSRC funded ARCHIE-WeSt HPC [EPSRC grant EP/K000586/1].

9 DATA ACCESS STATEMENT

The data sets, including all computed features, the evolved policies, and their performances, and the visualisations for all feature sets, are available from http://hdl.handle.net/11667/109.

REFERENCES

[1] Electoral Council of Australia. Electoral systems, 2008.
[2] T. Abell, Y. Malitsky, and K. Tierney. Features for exploiting black-box optimization problem structure. In *Revised Selected Papers of the 7th International Conference on Learning and Intelligent Optimization - Volume 7997*, LION 7, pages 30–36, New York, NY, USA, 2013. Springer-Verlag New York, Inc.
[3] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer. Feature based algorithm configuration: A case study with differential evolution. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, editors, *Parallel Problem Solving from Nature – PPSN XIV*, pages 156–166, Cham, 2016. Springer International Publishing.
[4] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer. Per instance algorithm configuration of cma-es with limited budget. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 681–688, New York, NY, USA, 2017. ACM.
[5] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 313–320, New York, NY, USA, 2012. ACM.

[6] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, 2016.

[7] J. Ceberio, A. Mendiburu, and J. A. Lozano. Are we generating instances uniformly at random? In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1645–1651, June 2017.

[8] C. García-Martínez, F. J. Rodriguez, and M. Lozano. Arbitrary function optimisation with metaheuristics. *Soft Computing*, 16(12):2115–2133, 2012.

[9] N. Hansen, A. Auger, D. Brockhoff, D. Tušar, and T. Tušar. Coco: Performance assessment. *arXiv preprint arXiv:1605.03560*, 2016.

[10] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1689–1696. ACM, 2010.

[11] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, 2009.

[12] D. S. Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms. In *5th and 6th DIMACS Implementation Challenges*. American Mathematical Society, 2002.

[13] B. Lacroix, L. A. Christie, and J. A. W. McCall. Interpolated continuous optimisation problems with tunable landscape features. In *Companion Proceedings GECCO (forthcoming)*. 2017.

[14] W. B. Langdon and R. Poli. Evolving problems to learn about particle swarm optimisers and other search algorithms. *IEEE Transactions on Evolutionary Computation*, 11(5):561–578, Oct. 2007.

[15] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Sttzle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, sep 2016.

[16] J. A. W. McCall, L. A. Christie, and A. E. I. Brownlee. Generating easy and hard problems using the proximate optimality principle. *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference - GECCO Companion fi15*, 2015.

[17] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 829–836, New York, NY, USA, 2011. ACM.

[18] O. Mersmann, M. Preuss, H. Trautmann, B. Bischl, and C. Weihs. Analyzing the bbob results by means of benchmarking concepts. *Evolutionary Computation*, 23(1):161–185, 2015. PMID: 24967695.

[19] M. A. Muñoz, M. Kirley, and S. K. Halgamuge. A meta-learning prediction model of algorithm performance for continuous optimization problems. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, pages 226–235, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[20] M. A. Muñoz and K. A. Smith-Miles. Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary computation*, 2016.

[21] M. A. Muoz, M. Kirley, and S. K. Halgamuge. Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, 19(1):74–87, Feb 2015.

[22] N. Pillay. Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *Journal of the Operational Research Society*, 63(1):47–58, 2012.

[23] R. Santana, A. Mendiburu, and J. A. Lozano. Multi-objective nm-landscapes. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 1477–1478, New York, NY, USA, 2015. ACM.

[24] K. Smith-Miles and S. Bowly. Generating new test instances by evolving in instance space. *Computers & OR*, 63:102–113, Nov 2015.

[25] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[26] R. Tanabe and A. Fukunaga. On the effect of computational budgets for tuning differential evolution: A parameter study. *Transaction of the Japanese Society for Evolutionary Computation*, 6(2):67–81, 2015.

[27] R. Tanabe and A. Fukunaga. Tuning differential evolution for cheap, medium, and expensive computational budgets. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2018–2025, May 2015.

[28] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaskowski, U.-M. O'Reilly, and S. Luke. Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, Mar. 2013.

[29] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[30] J. R. Woodward and J. Swan. Template method hyper-heuristics. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1437–1438, New York, NY, USA, 2014. ACM.