Statistical comparison in empirical computer science with minimal computation usage

Timothée Mathieu, Philippe Preux

SHORT: Statistical comparison in empirical computer science with minimal computation usage

Timothée Mathieu
timothee.mathieu@inria.fr
Inria, Université de Lille, CNRS, UMR 9189 CRIStAL
Villeneuve d'Ascq, France

Philippe Preux
philippe.preux@inria.fr
Université de Lille, CNRS, Inria, UMR 9189 CRIStAL
Villeneuve d'Ascq, France

ABSTRACT

The replicability of computational experiments remains a fundamental question. For example, the machine learning community has recently become aware of the poor replicability of many experimental studies that aim at comparing the performance of various algorithms. Due to computational costs, it is often necessary to use methods that require as few computations as possible to obtain a replicable conclusion. The conclusion of the comparison should also be replicable which calls for appropriate statistical tests. AdaStop is a recently introduced statistical test based on multiple group sequential tests. AdaStop adapts the number of executions of each experiment to stop as early as possible while ensuring that enough information is available to distinguish algorithms that perform better than the others in a statistically significant way. AdaStop has been initially exemplified on reinforcement learning tasks. In this short paper, we consider 3 case studies to investigate the use AdaStop beyond its original field of application, and demonstrate that it is a test that may be used on a wide range of application domains.

CCS CONCEPTS

• **Mathematics of computing** → *Nonparametric statistics*; • **General and reference** → **Empirical studies**; • **Computing methodologies** → *Sequential decision making*.

KEYWORDS

Statistical Reproducibility, Statistical Tests, Benchmarking

ACM Reference Format:
Timothée Mathieu and Philippe Preux. 2024. SHORT: Statistical comparison in empirical computer science with minimal computation usage. In *ACM Conference on Reproducibility and Replicability (ACM REP '24), June 18–20, 2024, Rennes, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3641525.3663618

1 INTRODUCTION

In this paper, we are interested in the experimental comparison of the performances of two or more programs solving the same problem. Our goal is that this comparison is reproducible with statistical guarantees. The *statistical reproducibility* of a computational experiment, also known as *replicability*[1], is the fact that two different groups of people reach the same qualitative conclusion using their own implementation of an algorithm, their own sets of data, their own computational resources, etc., up to a certain statistical level of confidence α.

Often, the performance measure is quantified by a real number that quantifies efficiency. There are many sources of randomness that make the performance measure vary from one experiment to the other. Depending on how we measure the performance of a program, sources of randomness range from the workload of the computer on which the experiments are run, network communication if a network is involved, the efficiency of the code generated by the compiler, down to the generating process of datasets, in particular in Machine Learning.In order to measure the uncertainty in the performance measure and to assess whether this uncertainty prevents us from concluding about the superiority of one algorithm on some others, we use statistical methods. Statistical methods are based on the repeated execution of an experiment, which is costly. To minimize the cost while assuring statistical reproducibility, we have recently introduced a new statistical test named AdaStop [8]. The design of AdaStop is closely related to the so-called "replicability crisis" [7] and the misuse of statistical tests for validating results with ad-hoc methods without proper theoretical background [5]. AdaStop was developed in the context of machine learning where the execution of a single experiment may take a long time (days, weeks). Aside the theoretical properties of AdaStop, we showed how this new statistical test may indeed be used to compare several competing algorithms, rank them according to their performance, while minimizing the number of executions, hence minimizing the total computational effort. In this paper, our goal is to investigate the use of AdaStop in various domains: energy consumption, global optimization, and machine learning. AdaStop is unique: it is the first statistical hypothesis test that addresses this particular type of decision problem, being non parametric (agnostic to the distribution of the data), and adaptive (aiming at minimizing the number of samples needed to make a decision). Aside the theoretical grounding of AdaStop, we provide an open source implementation which is very simple to use.

We address the following research questions:

(1) Is AdaStop useful beyond the domain of reinforcement learning?
(2) Can we use AdaStop to reproduce already published results, bringing them statistical guarantees?

[1]We are targetting here replicability as defined by the ACM in https://www.acm.org/publications/policies/artifact-review-and-badging-current and [11].

ACM REP '24, June 18–20, 2024, Rennes, France

ACM ISBN 979-8-4007-0530-4/24/06
https://doi.org/10.1145/3641525.3663618

2 ADASTOP FOR STATISTICAL REPRODUCIBILITY

Let us first define 2 important notions.

In a computational experiment, we compare neither algorithms, nor programs. We compare programs, each with a certain set of values for its hyperparameters[2]. Aiming at using a precise wording, we introduce the notion of a *fully specified experimental design* (shortened to FSED) to be a program along the specification of the value of its hyperparameters, and the particular task it is solving. The *score of an FSED* results from one execution of an FSED. This is a numerical value used to compare the performance of a set of FSEDs. A score measures the criterion we want to compare a set of FSEDs upon. Given a maximum number of scores ("budget"), AdaStop decides how many scores are needed to conclude on the equality or difference of the mean scores of a set of FSEDs. In particular, AdaStop is designed to be effective when working with very few scores in order to make a statistically significant decision at a minimal computational cost. Using AdaStop to compare two FSEDs, there are two possible outcomes: equal or different. If the outcome is "different" (we will denote this $a \prec b$ meaning that a is smaller than b), we know that the mean of the score of one of the two FSEDs (b) is larger than the other one (a). If the outcome is "equal" (we will denote this $a \equiv b$), this means either that the maximum budget was not large enough to distinguish between the two FSEDs, or that the two FSEDs are really not distinguishable and the variability of their performance does not allow to conclude that one performs better than the other. AdaStop uses a Group Sequential Test approach to answer a multiple hypothesis test problem. In its simplest form, to compare two FSEDs, AdaStop works as follows: for N and K two integers:

- until either K rounds of test have been done or the test stops, do the following steps:
  (1) collect N scores from each FSED.
  (2) Compute the test statistic for each FSED using these N scores and all scores collected beforehand.
  (3) If the test statistic is satisfied, stop and conclude that the two FSEDs are different, otherwise go to step (1).
- Return the decision of the test.

In the case where there are more than two FSEDs, the statistical test is a "multiple hypothesis test". This means that in addition to the error we get due to the sequential estimation, we also inevitably accumulate the errors done for each comparison of two FSEDs. We have to take this into account when designing the test for more than two algorithms. In this case, AdaStop controls an alternative test error metric (Family-Wise Error, FWE for short) that consists in the probability of making at least one false discovery when testing all the couples of FSEDs regarding their equality/inequalities. AdaStop has been shown to have a strong control on the type I error in the case of comparison of 2 FSEDs and to control the FWE for more than 2 FSEDs.We note α this error (either FWE for multiple test or type I error for a two sample test). AdaStop is nonparametric which means that it does not make any strong assumption on the distributions of scores, i.e. the scores do not need to be Gaussian, but they still need to concentrate around their mean so that the mean is a good measure of central behavior for the scores. All these aspects of AdaStop are detailed in [8].

[2]we use this term with its meaning in machine learning where the word "hyperparameter" refers to parameters which values are set by the user whereas a "parameter" gets its value from the data it is trained upon: for instance, the number of neurons in a neural network is usually a hyperparameter, whereas their weights are parameters.

3 APPLICATIONS OF ADASTOP

In this section, we apply AdaStop to three cases studies:

Energy consumption of programming languages in which we compare the energy consumption to compute the solution of an N-body problem in double precision with different programming languages.

Global optimization in which we compare the value obtained by several optimization algorithms on a non-convex objective function.

Machine Learning in which we compare a set of neural network weight update rules to study the impact of a certain hyperparameter (weight decay) on a supervised classification task.

We took care of using Guix [1, 2] as much as possible in order to maximize the reproducibility. The global optimization and the energy consumption case studies are using Guix and run on CPU. On the other hand, the machine learning study runs on GPU and some packages are missing in Guix. In this case, we used Benchopt[10] which depends on Conda to have a result that is as reproducible as possible. All the scores of all these experiments are available on https://gitlab.inria.fr/tmathieu/adastop_acm in `csv` files in the `data` folder. These scores can be used with AdaStop in a completely reproducible manner to obtain the results presented in this paper. This comparison is bitwise reproducible when using AdaStop through Guix. AdaStop is open source and freely available at https://github.com/TimotheeMathieu/adastop. The 3 case studies are meant to demonstrate our methodology based on AdaStop. Our goal is absolutely not to obtain new results, but rather we hope that our methodology will confirm the previously published experimental results. In this case, the already published results will be strengthened by gaining statistical significance. Moreover, if the confirmation succeeds, we may show that the same result could have been obtained with fewer computations, then at a smaller computational cost, and by reducing the emission of pollution. Running the energy consumption task took around 1 hour on an i9 laptop running Linux/Archlinux, the optimization task took around 15mn on the same laptop, and the machine learning task took around 24h on a server equipped with GPUs, running Linux/Ubuntu.

3.1 Energy consumption of programming language study

The first case study consists in comparing the energy consumption of a set of programs written in different programming languages. All these programs implement the same algorithm that computes the solution of an N-body problem in double precision. The task and implementation are taken from [12]. The energy consumption was measured with the RAPL [4] Intel tool, using the `perf` Linux command line (this is different from [12] where the authors used RAPL library in C to compute the energy consumption). We took care to have as few processes as possible running at the same time on the computer on which these measurements were done. In particular, the runs were done on a laptop, not on a server. First,

we used AdaStop to check that Guix containers do not impact the energy consumption, at least for the C language. To do this, we compiled the N-body C program with Guix and compared the energy consumption of running the resulting binary either through a Guix container or without a Guix container. We set the $K = 10$, $N = 5$, and $\alpha = 0.01$ for each of the two tasks, "C" and "C-Guix". The result was that AdaStop used all the budget to decide that "C" and "C-Guix" have the same consumption (the mean consumption of "C" was 47.39 Joules with standard deviation 5.02, while the mean consumption of "C-Guix" was 46.41 Joules with standard deviation 4.30). Knowing that Guix containers do not impact the power consumption significantly on this test problem (we suppose that this is true for all languages even though we only did the test for C), we evaluated the proposed N-body benchmark on the following languages: C, C++, Fortran, Julia, Lisp, Pascal, Python (using PyPy). If the program could be compiled, we compiled the source code before running it using the same compiler options as in the original publication [12]. The power consumption is measured only on the running FSEDs: compilation cost is not measured as it is very small in regard to the running cost.

Parameters for AdaStop: we set $K = 10$, $N = 5$, $\alpha = 0.01$. AdaStop concludes:

$$\text{C++} \equiv \text{C} \prec \text{Pascal} \prec \text{Fortran} \equiv \text{Julia} \equiv \text{Lisp} \prec \text{Python}.$$

The numerical figures can be found in Tables 1 and 2.

FSED 1	FSED 2	mean diff.	std FSED 1	std FSED 2
Python	Lisp	403.71	25.89	6.17
Lisp	Julia	1.65	6.17	4.54
Julia	Fortran	0.79	4.54	7.85
Fortran	Pascal	6.83	7.85	7.14
Pascal	C	23.83	7.13	4.51
C	C++	0.11	4.51	4.45

Table 1: Table of power consumption comparison of 7 programming languages (in Joules). Each row provides the figures regarding a pair of FSEDs. The first two columns contain the names of the two FSEDs (here named after the programming language). The third column contains the mean difference between the power consumptions of the two FSED. The fourth and fifth columns contain the standard deviations of power consumption for each of the two FSEDs in the row.

Language	C	C++	Pascal	Fortran	Julia	Lisp	Python
Nb. scores	50	50	20	50	50	50	20

Table 2: Number of scores obtained before AdaStop concludes.

Table 2 shows that Pascal and Python programs could be ranked in a statistically significant manner using only 20 runs whereas the other languages required the use of the whole budget. This is quite a lot of scores compared to the 10 scores used in the original study [12]. This is due to the fact that we want to be fairly confident of our results, and we set $\alpha = 0.01$ and a maximum of $KN = 50$ scores collected for each FSED. By choosing a less conservative (larger) value of α, we would need much less scores: for instance, with the usual $\alpha = 0.05$ and $K = 5, N = 5$, we can re-execute the experiment and obtain the same conclusions using only these 25 scores for every language except for Python and Pascal for which AdaStop makes a decision with only 5 scores (the power of the resulting test is smaller as K is smaller). The reader may find it weird that we could decide with only 5 runs in this case while we reported being able to decide after 20 runs only above. This is due to complex, non-linear, interactions between the parameters K, N, and α. Please note that we did not use the same version of compilers or interpreters as in the original publication [12] because 1) we are only trying to use this as a case study, and we are not trying to reproduce the original results 2) even with the same versions, we may not obtain the same ranking[3] as in [12] because the experiments have been performed on very different computers.

3.2 Global minimization study

The second case study concerns a global minimization problem taken from [3]. We are looking for the optimum of the F_4 function from the benchmark [13] using three different algorithms: the CMA-ES implementation in the python `cma` library [6], the `scikit-opt`[4] implementation of the particle swarm algorithm (PSO), and the `scipy` [14] implementation of differential evolution (DE).

Parameters for AdaStop: we set $K = 10$, $N = 10$, and $\alpha = 0.01$. After computing 30 scores[5] for each FSED, AdaStop concludes:

$$\text{CMA} \prec \text{DE} \prec \text{PSO}.$$

The numerical results can be found in Tables 3. The mean score for PSO is very large compared to the others and this is mainly due to a few outliers: occasionally, PSO did not reach an area close to the global minimum and had a very large error. However, it still took 30 scores for AdaStop to conclude for sure about inequality because these outliers also caused PSO to have a large variability in its score. This large variability led AdaStop to need a large number of scores to conclude about the mean performance of PSO.

FSED 1	FSED 2	mean diff.	std FSED 1	std FSED 2
PSO	DE	65.23	140.23	5.37e-07
DE	CMA	1.37e-06	5.37e-07	2.84e-14

Table 3: AdaStop results for the global minimization case study reporting the value of the minimum found by each FSED.

3.3 Machine learning study

In this section, we use AdaStop in the context of machine learning. For this case study, we use the Benchopt [9] Python library to compare the effect of weight-decay on the training of a ResNet18 neural network for image classification on CIFAR-10. In this task, 1) scores have a large variability, and 2) each run is computationally costly.

[3] In [12], the ranking obtained was Fortran ≺ C++ ≺ C ≺ Pascal ≺ Lisp ≺ Python. and Julia was not ranked at the time.
[4] https://scikit-opt.github.io/
[5] The initial study in [3] used 50 repetitions but it is not really comparable because they compared a larger number of FSED.

Parameters for AdaStop: due to the computational cost for this application, we set $K = 5$, $N = 6$, and $\alpha = 0.05$.

Adam and SGD weight update rules are compared, each with different values for the weight decay. We denote A for Adam and S for SGD, indexed with the weight decay parameter. AdaStop concludes on the following ranking:

$$A_{0.002} \equiv A_{0.02} \prec S_{5e-05} \prec A_{0.2} \prec A_{0.5} \prec S_{0.001} \prec S_{0.005}.$$

The numerical results can be found in Tables 4 and 5. Although the difference of score between 2 FSEDs can be pretty small in this study, AdaStop was still able to conclude in most cases because the variability was small. The randomness comes from the initialization of the pseudo-random number generator, the initialization of the weights of the neural network.

FSED 1	FSED 2	mean diff.	std FSED 1	std FSED 2
$S_{0.005}$	$S_{0.001}$	0.017	5.9e-3	9.4e-4
$S_{0.001}$	$A_{0.5}$	2.1e-3	9.4e-4	5.2e-4
$A_{0.5}$	$A_{0.2}$	1.8e-3	5.2e-4	2.5e-4
$A_{0.2}$	S_{5e-5}	1.1e-3	2.5e-4	1.6e-4
S_{5e-5}	$A_{0.02}$	1.9e-4	1.6e-4	9.4e-5
$A_{0.02}$	$A_{0.002}$	2.8e-5	9.4e-5	1.1e-4

Table 4: Table of results of AdaStop on the machine learning case study reporting the test errors of FSED.

Algorithm	Adam				SGD		
Weight decay	0.5	0.2	0.02	0.002	0.005	0.001	5e-05
Nb scores	6	6	30	30	6	6	12

Table 5: Number of scores needed for AdaStop to conclude.

3.4 Discussion about AdaStop parameters

We would like to discuss the choice of the parameters K, N, and α. Regarding α, the traditional value in statistics is $\alpha = 0.05$ which means that we accept a risk of a wrong rejection of the null hypothesis[6] of 5%. The smaller α, the more conservative the test, the less risk we allow to take, and consequently, the more samples are needed. So, this is a matter of a trade-off. In the first case study where we assess the energy consumption overhead due to the use of Guix, we expect that this is negligible, so we take a small α to be more confident about the conclusion of the test. N may be chosen according to the level of concurrency of the computer we run the experiment on. Indeed, the N runs of a given FSED may be done in parallel at no extra time cost, but with improved confidence in the test conclusion and efficiency in its execution. Moreover, if a single run of an FSED takes a short amount of time, we may allow ourselves to perform more runs of each FSED even if they are not parallelized. This is the case in the second case study where each single run is fast, hence our choice of $N = 10$. Regarding K, our choice is pragmatic: in the third case study, a single run of an FSED takes much more time than in the other experiments, so we try to balance the total amount of time the experiment may take if it consumes the whole budget and the accuracy of the test. Beyond these intuitions, we would like to raise the fact that the value of K controls the type II error, that is the statistical power of the test. For the moment, the theory behind it is not yet established, and we are currently working on it. Once we have this theory, we will be able to provide strong guidelines for the choice of K. Please note that this is a classical limitation in nonparametric hypothesis testing and practitioners often resort to heuristics (e.g. assume Gaussian model, use asymptotic result, etc) to set a value for the maximal sample size in a hypothesis test. Let us also mention that the theory behind AdaStop requires the setting of K before running the test. AdaStop is adaptive in the sense that it decides when to stop given that it will not perform more than K rounds of test. Relaxing the constraint of setting K corresponds to another type of algorithms (sequential tests) that we are also currently investigating. The main challenge here is to develop its theory for very small samples of data.

[6] In our context, rejection of null hypothesis means to decide the mean scores are different even though in reality they are the same.

4 DISCUSSION

We address the 2 research questions raised above:

(1) Can we use AdaStop beyond the domain of reinforcement learning?

Section 3 answers positively. We were able to straightforwardly apply AdaStop to three different domains of computer science, exhibiting different characteristics.

(2) Can we use AdaStop to reproduce already published results, then bringing statistical guarantees?

We found that AdaStop is not the source of any problem to reproduce already published results. When we fail to reproduce earlier results, this is due to a lack of availability of the exact experimental environment. To give a precise example, in the first case study, AdaStop concludes that C++ and C can not be statistically distinguished whereas this point was not very clear in the original publication; moreover, Fortran was ranked as the language consuming the less energy while our experiments rank it differently (and in a rather surprising way).

5 CONCLUSION

In a prior work, we introduced AdaStop which is a new statistical hypothesis test designed to compare the performance of a set of programs on a given task. AdaStop is a way to increase the replicability of computational experiments. In this paper, we investigate the use of AdaStop in 3 different domains, revisiting previously published works and strengthening their conclusions by adding statistical guarantees about their conclusions, at a rather minimal computational cose. The main direction for future work is to develop further the theory to control the statistical power of AdaStop (type II error).

Acknowledgements

Ph. Preux acknowledges the financial support of the Métropole Européenne de Lille (MEL), ANR, Inria, Université de Lille, through the AI chair Apprenf number R-PILOTE-19-004-APPRENF. Both authors acknowledge the Scool research group for an outstanding working environment.

REFERENCES

[1] Ludovic Courtès. 2013. Functional Package Management with Guix. In *European Lisp Symposium*. https://arxiv.org/abs/1305.4584

[2] Ludovic Courtès and Ricardo Wurmus. 2015. Reproducible and user-controlled software environments in HPC with Guix. In *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*. Springer, 579–591.

[3] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation* 1, 1 (2011), 3–18.

[4] Dimitrov, Martin and Strickland, Carl and Kim, Seung-Woo and Kumar, Karthik and Doshi, Kshitij. [n. d.]. *Intel® Power Governor*. https://software.intel.com/en-us/articles/intel-power-governor

[5] Andrew Gelman and Eric Loken. 2016. The statistical crisis in science. *The best writing on mathematics (Pitici M, ed)* (2016), 305–318.

[6] Nikolaus Hansen, Yoshihikoueno, ARF1, Gabriela Kadlecová, Kento Nozawa, Luca Rolshoven, Matthew Chan, Youhei Akimoto, Brieglhostis, and Dimo Brockhoff. 2023. CMA-ES/pycma: r3.3.0. https://doi.org/10.5281/ZENODO.7573532

[7] Eric Loken and Andrew Gelman. 2017. Measurement error and the replication crisis. *Science* 355, 6325 (2017), 584–585. https://doi.org/10.1126/science.aal3618 arXiv:https://www.science.org/doi/pdf/10.1126/science.aal3618

[8] Timothée Mathieu, Riccardo Della Vecchia, Alena Shilova, Matheus Centa de Medeiros, Hector Kohler, Odalric-Ambrym Maillard, and Philippe Preux. 2023. AdaStop: sequential testing for efficient and reliable comparisons of Deep RL Agents. arXiv:2306.10882 [cs.LG] submitted.

[9] Thomas Moreau, Mathurin Massias, Alexandre Gramfort, Pierre Ablin, Pierre-Antoine Bannier, Benjamin Charlier, Mathieu Dagréou, Tom Dupre la Tour, Ghislain Durif, Cassio F Dantas, et al. 2022. Benchopt: Reproducible, efficient and collaborative optimization benchmarks. *Advances in Neural Information Processing Systems* 35 (2022), 25404–25421.

[10] Th. Moreau, M. Massias, A. Gramfort, P. Ablin, P-A. Bannier, B. Charlier, M. Dagréou, T. Dupré la Tour, G. Durif, C. F. Dantas, Q. Klopfenstein, J. Larsson, E. Lai, T. Lefort, B. Malézieux, B. Moufad, B. T. Nguyen, A. Rakotomamonjy, Z. Ramzi, J. Salmon, and S. Vaiter. 2022. Benchopt: Reproducible, efficient and collaborative optimization benchmarks. In *NeurIPS*. https://arxiv.org/abs/2206.13424

[11] National Academies of Sciences, Engineering, and Medicine. 2019. *Reproducibility and Replicability in Science*. The National Academies Press.

[12] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy efficiency across programming languages: how do energy, time, and memory relate?. In *Proceedings of the 10th ACM SIGPLAN international conference on software language engineering*. 256–267.

[13] Ponnuthurai N Suganthan, Nikolaus Hansen, Jing J Liang, Kalyanmoy Deb, Ying-Ping Chen, Anne Auger, and Santosh Tiwari. 2005. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. *KanGAL report* 2005005, 2005 (2005), 2005.

[14] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2