# On Equivalence of Algorithm's Implementations: The CMA-ES Algorithm and Its Five Implementations

Rafał Biedrzycki
Institute of Computer Science, Warsaw University of Technology
Warsaw, Poland
rbiedrzy@elka.pw.edu.pl

## ABSTRACT

When a new optimization algorithm is proposed, it is compared with state-of-the-art methods. That comparison is made using implementations of the algorithms, but names and versions of the implementations are usually not revealed. This paper compares five implementations of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) taken from a trusted source. The comparisons were performed using the Comparing Continuous Optimizers (COCO) platform. The results show that all examined implementations produce a different outcome. The variation of the results stems from differences in the auxiliary codes of the implementations and from implementing an algorithm which is still under development. It is therefore important to use an appropriate implementation for experiments. Using a weak implementation can lead to the wrong conclusions.

## CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**;

## KEYWORDS

experiments replication, benchmarking, algorithm-implementation gap, CMA-ES

**ACM Reference Format:**
Rafał Biedrzycki. 2019. On Equivalence of Algorithm's Implementations: The CMA-ES Algorithm and Its Five Implementations. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3322011

## 1 INTRODUCTION

The reproducibility of an experiment is a key aspect of scientific research. It was noticed [2] that in many articles, experiments are performed and the results are compared in a way that is not scientifically correct. The authors of the above work proposed a set of guidelines that stress the need for providing a description of the implementation and specification of the algorithm's parameters and stopping conditions. Poor experimental research methodology was also criticised in [5]. The authors pointed out that many articles do not define the scope of claims of the superiority of an introduced algorithm. It was also noticed that in many papers, the level of detail does not allow the algorithm to be reimplemented. In [11], common pitfalls in comparison of the outcome of experiments are identified. The paper provides guidelines and a checklists which should be followed before an article is published.

All aforementioned contributions do not describe what to do when many implementations of the algorithm are available. Usually, important algorithms have many implementations in popular languages. For the R language, several implementations of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [9] and two implementations of Differential Evolution [10] were compared in [3]. The results showed that there are differences in the outcome of different implementations of the same algorithm. The results can be perceived as not interesting by the optimization community because R is not a typical optimization environment and the examined R packages are perceived as not trusted sources of implementations since they are not made by the authors of the algorithms. The aim of this paper is to show that differences between implementations also exist when implementations come from a trusted source.

CMA-ES is used here because it is an important optimization algorithm with many high-quality implementations made in several programming languages by one of its authors.

## 2 EXPERIMENTAL SETUP

The experiments were performed using version 2.2.1.10 of the Comparing Continuous Optimizers (COCO) platform [1] using 24 noiseless functions (BBOB'2009). COCO provides benchmark functions, the standard, automatized benchmarking procedure and tools for processing and visualizing data. It uses the concept of average runtime (aRT) that is required to achieve a certain error level. The statistical significance of results is tested with the rank-sum test with Bonferroni correction. COCO is typically used in unconstrained searches, but it defines functions that return the bounds of the area of interest. These functions are used to generate an initial solution and they are used in COCO's exemplary experiment files to run optimizers that perform bound constrained searches. These bounds are also used in this contribution to better reflect a real-world application of the optimization algorithm, where bounds usually stem from physics. For all implementations being compared, the budget was set to $5 \cdot 10^7$ and initial step size ($\sigma$) was set to $0.3(u - l)$, where $u$ is the upper and $l$ is the lower bound of the parameters' value. If an algorithm stops before exploiting its given budget, it is restarted. The first starting point is defined by COCO, and the starting points for restarts are generated by COCO's function *initial_solution_proposal*.

*GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic*

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3322011

**Table 1: The average runtime (aRT) divided by the best aRT measured during BBOB-2009 at the target error level $10^{-7}$ in 5-D as returned by COCO.**

| Fun. | C | Java | Matlab | Python | Py. sim. | Py. sq. pen. |
|---:|---:|---:|---:|---:|---:|---:|
| f1 | 115 (10) | 105 (7) | 107 (6) | 109 (9) | 117 (13) | **104** (12) |
| f2 | 49 (4) | 46 (3) | 43 (3) | **29** (2) | 46 (5) | 30 (5) |
| f3 | 379 (332) | **283** (274) | 340 (419) | 600 (798) | 471 (388) | 327 (220) |
| f4 | 6982 (6577) | 6723 (7326) | 9849 (1e4) | 5972 (9309) | 5681 (1e4) | **3874** (4264) |
| f5 | 172 (29) | 435 (39) | 47 (22) | 146 (23) | 97 (43) | **33** (12) |
| f6 | 2.6 (0.3) | **2.3** (0.4) | 2.4 (0.5) | 2.4 (0.3) | **2.3** (0.2) | 2.4 (0.3) |
| f7 | 14 (14) | 7.1 (8) | 10 (11) | **2.9** (2) | 8.3 (12) | 3.0 (4) |
| f8 | 12 (7) | 10 (1) | 12 (6) | 13 (3) | 10 (1) | **9.5** (5) |
| f9 | 12 (2) | 12 (3) | 12 (5) | 13 (11) | 13 (5) | **10** (1) |
| f10 | 8.3 (4) | 5.3 (0.6) | 4.8 (0.6) | 4.4 (2) | 4.8 (0.6) | **3.3** (0.4) |
| f11 | 3.0 (1) | 2.7 (0.2) | 2.5 (0.1) | 2.1 (1) | 2.6 (0.2) | **1.5** (0.1) |
| f12 | 6.7 (5) | 5.7 (3) | 4.2 (3) | 6.3 (3) | 6.2 (3) | **3.4** (2) |
| f13 | 3.4 (0.5) | 3.6 (0.4) | 3.1 (0.4) | **2.0** (0.2) | 2.9 (0.3) | 2.2 (1) |
| f14 | 10 (1) | 10 (1) | 8.3 (0.8) | 5.1 (0.4) | 9.1 (1) | **4.9** (0.8) |
| f15 | **20** (10) | **20** (27) | 43 (44) | 22 (36) | 24 (26) | 34 (28) |
| f16 | 25 (12) | 5.7 (8) | 16 (20) | 12 (17) | **2.7** (4) | 7.9 (6) |
| f17 | 39 (52) | **22** (17) | 50 (42) | 42 (46) | 40 (61) | 59 (102) |
| f18 | 440 (311) | 501 (659) | 730 (549) | 137 (104) | 265 (182) | **134** (205) |
| f19 | 355 (350) | 386 (357) | 288 (180) | 117 (50) | 382 (379) | **86** (53) |
| f20 | 105 (71) | 60 (50) | 101 (104) | 49 (89) | 49 (56) | **35** (44) |
| f21 | 15 (18) | **7.1** (5) | 12 (8) | 14 (21) | 13 (16) | 18 (20) |
| f22 | 38 (62) | 46 (65) | 54 (55) | 57 (68) | **34** (30) | 42 (92) |
| f23 | 49 (79) | **4.8** (3) | 16 (13) | 6.4 (6) | 5.8 (2) | 7.5 (11) |
| f24 | ∞ | ∞ | **60** (60) | 62 (56) | ∞ | ∞ |
| Σ best | 1 | 6 | 1 | 3 | 3 | 12 |

The CMA-ES implementations were downloaded from Hansen's homepage [6]. The exact versions of used packages, as extracted from the source code, are as follows: Java – "0.99.40", C – "3.20.00.beta", Python purecma – "3.0.0", Python – "2.6.0, revision 4423", Matlab – "3.33.integer". The Python implementation includes two variants of Bound Constraint Handling Methods (BCHMs): the coordinate transformation version [7] and weighted quadratic penalty [8]. Both versions were included in the experiments because the C implementation of CMA-ES uses the transformation, and the Matlab version uses the penalty approach. The Java version uses resampling and the simplified CMA-ES version comes without constraint handling. To be able to use it the implementation was enriched by a simple additive quadratic penalty. More information on BCHMs can be found in [4]. The heuristic of setting population sizes $\mu$ and $\lambda$ was identical in all considered implementations so it was left untouched.

Since the aim of the paper is to check whether there are differences between implementations, the experiments were performed in five dimensions only.

## 3 RESULTS AND DISCUSSION

The aRTs divided by the best aRT measured during BBOB-2009 for all functions at the target error level $10^{-7}$ are shown in Table 1. It can be observed, that there are clear differences between the outcomes of all official implementations. The experiments showed that the best is the Python implementation with the weighted quadratic penalty (Py. sq. pen.), whereas the same implementation with default bound constraint handling (Python) is in joint third place with the Python simple (Py. sim.) implementation. The Java implementation is in second place. The Matlab and C versions are in last place.

The results of the official Python implementation coupled with two different BCHMs showed that BCHMs are a strong source of differences in the results. To further investigate the sources of the differences, implementations with the same BCHM were compared in pairs. Still, there were strong differences, i.e., for methods with a BCHM based on penalty (Matlab and appropriately configured Python), strong statistically significant differences were detected by COCO for functions 2, 8, 10, 11, 13, 14; and when comparing methods based on transformations of the parameters (Python and C), statistically significant differences were found for functions 2, 5, 10, 11, 13, 14, 23. These findings show that besides bound constraint handling, there are other differences between implementations that result in statistically significant differences in the results.

The inspection of the source code of the implementations reveals that more recent implementations implement more recent (improved) concepts of the algorithm. Other sources of differences are hidden in the auxiliary code responsible for the detection of numerical instabilities and in the stopping criterion connected with detection of the inability of further improvement.

## 4 CONCLUSIONS

The outcomes of the official CMA-ES implementations differ substantially. Therefore, it is important which of them is used. To make an experiment reproducible, experimenters should report the name and version of the used implementation. To make valuable conclusions, the authors should use the most up-to-date CMA-ES implementation, i.e. the Python implementation. The default bound constraint handling cannot be perceived as absolutely the best.

The observed differences will probably be much larger in a CMA-ES typical usage scenario, where practitioners from other fields do not set the maximal number of iterations or step size $\sigma$.

## REFERENCES

[1] [n. d.]. *COCO (COmparing Continuous Optimisers)*. http://coco.gforge.inria.fr/

[2] Richard S. Barr, Bruce L. Golden, James P. Kelly, Mauricio G. C. Resende, and William R. Stewart. 1995. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics* 1, 1 (01 Sep 1995), 9–32. https://doi.org/10.1007/BF02430363

[3] Rafał Biedrzycki. 2018. Differences that make a difference: comparing implementations of selected optimization algorithms in R language. In *Proc.SPIE*, Vol. 10808. 10808 – 10808 – 12. https://doi.org/10.1117/12.2501381

[4] Rafał Biedrzycki, Jarosław Arabas, and Dariusz Jagodziński. 2018. Bound constraints handling in Differential Evolution: An experimental study. *Swarm and Evolutionary Computation* (2018). https://doi.org/10.1016/j.swevo.2018.10.004

[5] Agoston E Eiben and Márk Jelasity. 2002. A critical note on experimental research methodology in EC. In *2002 IEEE Congress on Evolutionary Computation*. IEEE, 582–587.

[6] Nikolaus Hansen. [n. d.]. The CMA Evolution Strategy. ([n. d.]). Retrieved Oct 14, 2018 from http://cma.gforge.inria.fr/cmaesintro.html

[7] Nikolaus Hansen. 2016. The CMA Evolution Strategy: A Tutorial. *CoRR* abs/1604.00772 (2016). arXiv:1604.00772

[8] Nikolaus Hansen, A. S. P. Niederberger, L. Guzzella, and P. Koumoutsakos. 2009. A Method for Handling Uncertainty in Evolutionary Optimization With an Application to Feedback Control of Combustion. *IEEE Transactions on Evolutionary Computation* 13, 1 (Feb 2009), 180–197. https://doi.org/10.1109/TEVC.2008.924423

[9] Nikolaus Hansen and A. Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. https://doi.org/10.1162/106365601750190398

[10] Rainer Storn and Kenneth Price. 1995. *Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces.* Technical Report. TR-95-012, ICSI.

[11] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2014. Replication and comparison of computational experiments in applied evolutionary computing: Common pitfalls and guidelines to avoid them. *Applied Soft Computing* 19 (2014), 161 – 170. https://doi.org/10.1016/j.asoc.2014.02.009