Benchmarking Multivariate Solvers of SciPy on the Noiseless Testbed

Konstantinos Varelas, Marie-Ange Dahito

Benchmarking Multivariate Solvers of SciPy on the Noiseless Testbed

Konstantinos Varelas[1,2] and Marie-Ange Dahito[1,3]

[1]Inria, CMAP, École Polytechnique, IP Paris
[2]Thales LAS France SAS
[3]PSA Group
firstname.lastname@inria.com

ABSTRACT

In this article we benchmark eight multivariate local solvers as well as the global Differential Evolution algorithm from the Python SciPy library on the BBOB noiseless testbed. We experiment with different parameter settings and termination conditions of the solvers. More focus is given to the L-BFGS-B and Nelder-Mead algorithms. For the first we investigate the effect of the maximum number of variable metric corrections used for the Hessian approximation and show that larger values than the default are of advantage. For the second we investigate the effect of adaptation of parameters, which is proved crucial for the performance of the method with increasing dimensionality.

CCS CONCEPTS

• **Computing methodologies → Continuous space search**;

KEYWORDS

Benchmarking, Black-box optimization, local solvers, SciPy, Differential Evolution

ACM Reference format:
Konstantinos Varelas[1,2] and Marie-Ange Dahito[1,3]. 2019. Benchmarking Multivariate Solvers of SciPy on the Noiseless Testbed. In *Proceedings of Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, July 13–17, 2019 (GECCO '19 Companion),* 9 pages.
https://doi.org/10.1145/3319619.3326891

1 INTRODUCTION

A quantified performance comparison of optimization solvers is an important task for algorithm selection. Methods based on fundamentally different approaches, e.g. gradient-based methods, model-building methods, stochastic search algorithms etc. can be beneficial in particular groups of problems. Thus, assessing their performance on a well-understood set of problems such as the BBOB Noiseless testbed [8] is particularly helpful in order to expose their advantages and drawbacks and better understand them. Moreover, it is useful to compare how particular implementations of such methods are evolved and improved over time.

In this article, a variety of solvers either in a black-box setting or in a gradient based setting with approximation of the gradients is benchmarked. In particular, multivariate optimization solvers from the latest version of the Python SciPy[1] library are compared, under default or modified parameter settings. A similar study for a previous version of SciPy, that benchmarked six solvers of the library under default parameters has been presented in [1], where the Basin Hopping [21] restart strategy was used within each independent restart. In this study we follow a policy where independent restarts are applied when the corresponding termination criteria are met, until a given budget of function evaluations is exhausted. Based on a preliminary experimentation, we choose proper parameter settings and termination conditions for some algorithms such that their performance is not deteriorated.

The contribution in comparison to [1] is threefold: for the common benchmarked solvers, we compare the different parameter settings and restart policies. Furthermore, complete data sets for all dimensions are included (in [1] the results were restricted to dimensions 2, 5 and 20) and three additional solvers are benchmarked.

2 BENCHMARKED ALGORITHMS AND THEIR PARAMETER SETTING

In order to investigate such effects and identify proper settings prior to the performance comparison of all solvers, experimentation was performed separately up to some extent, concerning in most, but not all, cases the termination tolerances in search and objective space. The following algorithms were benchmarked, where a star indicates those included in [1] as described above: Nelder-Mead*, Powell*, BFGS*, L-BFGS-B*, Conjugate Gradient*, Truncated Newton, Differential Evolution, COBYLA and SLSQP*.

In the case of the quasi Newton L-BFGS-B algorithm [12] for high dimensional optimization, reducing the tolerance in objective values (`ftol` parameter with default value 10^{-8}) can be of advantage, in particular for ill-conditioned functions and for the Attractive Sector function, as presented in Figures 1 and 2. This performance improvement becomes more significant with increasing dimensionality. Thus, in our experimentation it was set to the float machine precision[2] for very high accuracy. More importantly, the maximum number of variable metric corrections for the Hessian approximation has to be set carefully. The default value is 10 and experiments showed performance improvement for increasing values up to $2 \times D$,

[1]Version: 1.2.1
[2]Equal to $2.220446049250313 \times 10^{-16}$

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

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326891

D being the problem dimension, as illustrated in Figure 3. Thus it was set to this value in our comparison. Furthermore, the effect of decreasing the step length for the finite difference approximation of the gradient was investigated to some extent: decreasing the default value of this parameter (10^{-8}) can improve the performance on particular functions, such as the Ellipsoid, while it shows worse success ratio on others. A more detailed study was presented in [2] and in the following comparison it is set to its default value.

The Nelder-Mead [14] simplex method is tested both in its default setting and with adaptation of parameters to the dimensionality of the problem [4], controlled by the `adaptive` flag.

For the modified Powell's conjugate direction algorithm [16] [18], the parameter `ftol` was set to 10^{-15}. As for the termination tolerance in the search space, different values of `xtol` were tested in the set $\{10^{-2}, 10^{-3}, 10^{-5}, 10^{-6}\}$. Values larger than the default (10^{-4}) typically can make the solver faster only for the easiest targets, while smaller values can show an improved success rate for high budgets. In the following the default value was chosen.

The truncated Newton algorithm [15] [13] requires an estimation of the optimal f value. Since it always lies in $[-1000, 1000]$ [9] and in accordance to the black-box setting where no prior information is available for the function, we set this value to -1000.

The SLSQP method that uses Sequential Least-Squares Programming [11] has been tested for different values of `ftol` (10^{-6}, 10^{-9}, 10^{-12} and 10^{-15}). Same as L-BFGS-B and Powell's algorithm, it was sensitive to this parameter, that was set to 10^{-15} in the performance comparison.

The original BFGS [15] method, the conjugate gradient algorithm by Polak and Ribiere [15], the global optimization Differential Evolution [20] method as well as the Constrained Optimization BY Linear Approximation (COBYLA) algorithm [17] are benchmarked in their default setting.

In cases where the solver supported constraint handling, no constraints were applied. Finally, the maximum iterations were set to values large enough (wherever applicable), in order to avoid termination before convergence.

3 EXPERIMENTAL PROCEDURE

All solvers were run on the `bbob` testbed with restarts for a maximum budget of $10^5 \times D$ function evaluations (at minimum, for solvers that did not support a termination callback such as COBYLA). For all runs, the initial point was chosen uniformly at random in $[-4, 4]^D$ and with the function value evaluated at this point. In the special case of Differential Evolution where no initial point is given, the domain bounds were set as $[-5, 5]^D$.

4 CPU TIMING

The Python code was run on several multicore machines (not exclusively) with different number of cores. The time per function evaluation, measured in 10^{-5} seconds, for different dimensions along with the corresponding processor type is presented in Table 1.

Algorithm	Processor Type	2-D	3-D	5-D	10-D	20-D	40-D
Nelder-Mead	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.2	2.3	2.4	2.9	3.5	5.0
Adaptive Nelder-Mead	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.1	2.1	2.2	2.6	3.3	4.8
Powell	Intel Core Haswell, no TSX @ 2.29GHz	2.1	2.5	2.4	2.5	2.9	4.0
BFGS	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.7	2.7	2.6	2.7	3.3	5.2
L-BFGS-B	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz	2.9	2.3	2.1	2.3	3.1	5.4
Conjugate Gradient	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	2.2	1.9	1.4	1.3	1.9	3.8
Truncated Newton	Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz	1.4	1.4	2.5	2.0	2.8	5.0
Differential Evolution	Intel(R) Xeon(R) CPU X5650 @ 2.67GHz	8.4	8.4	8.5	9.3	11.0	14.0
COBYLA	Intel Core Haswell, no TSX @ 2.29GHz	0.51	0.53	0.65	0.96	2.1	8.2
SLSQP	Intel Core Haswell, no TSX @ 2.29GHz	2.0	1.9	1.8	2.1	1.9	2.2

Table 1: CPU timing per function evaluation

5 RESULTS

Results from experiments according to [10] and [5] on the benchmark functions given in [3, 8] are presented in Figures 4, 5, 6, 7, and 8. The experiments were performed with COCO [7], version 2.3, the plots were produced with version 2.3. The solvers benchmarked in [1] are denoted by a prefix "B-" in the corresponding name and the data were obtained by the data archive that COCO provides.

The **average runtime (aRT)**, used in the figures, depends on a given target function value, $f_t = f_{opt} + \Delta f$, and is computed over all relevant trials as the number of function evaluations executed during each trial while the best function value did not reach f_t, summed over all trials and divided by the number of trials that actually reached f_t [6, 19]. **Statistical significance** is tested with the rank-sum test for a given target Δf_t using, for each trial, either the number of needed function evaluations to reach Δf_t (inverted and multiplied by -1), or, if the target was not reached, the best Δf-value achieved, measured only up to the smallest number of overall function evaluations for any unsuccessful trial under consideration.

6 OBSERVATIONS AND CONCLUSION

Aggregated results over all 24 functions of the suite show the effectiveness of SLSQP. In dimension 5, it has the highest success rate for a budget range $[18D, 800D]$ while in dimension 20, it dominates all solvers up to $\sim 1400D$ function evaluations, after which it is outperformed by L-BFGS-B.

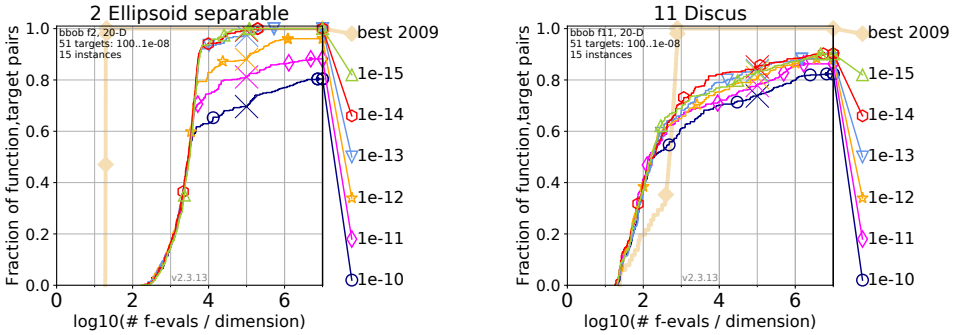

Figure 1: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ of the ill-conditioned separable Ellipsoid and Discus functions in 20-D for L-BFGS-B. The graphs correspond to different values of f-tolerance for termination.

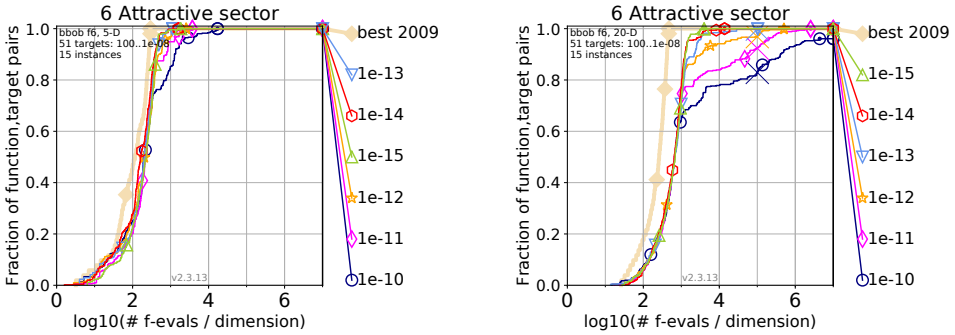

Figure 2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ of the Attractive Sector function in 5-D and 20-D for L-BFGS-B. The graphs correspond to different values of f-tolerance for termination.

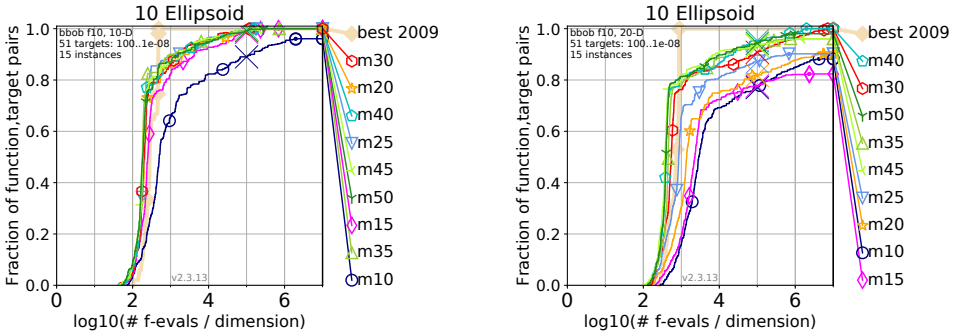

Figure 3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ of the ill-conditioned separable Ellipsoid function in 10-D and 20-D for L-BFGS-B. The graphs correspond to different values of maximum number of variable metric corrections for the Hessian approximation.

It is interesting to see the performance difference of SLSQP and B-SLSQP in unimodal functions such as the separable Ellipsoid function: in $5D$, the runtimes are almost equal for the easiest targets and then SLSQP is faster by an increasing factor, until termination criteria start to become effective. Performance differences between the early and recent implementation of SciPy, which are not due to the different parameter setting or restart policy of [1], are also observed for BFGS and Nelder-Mead, showing an improvement of the library implementation.

Comparing BFGS and L-BFGS-B, the latter can show better performance for some functions in *all* dimensions, as it is the case for the Sphere, Linear Slope, original and rotated Rosenbrock and Bent Cigar functions. Overall, the picture is more diverse: BFGS has same or higher success rate in the budget ranges $[25D, 125D]$

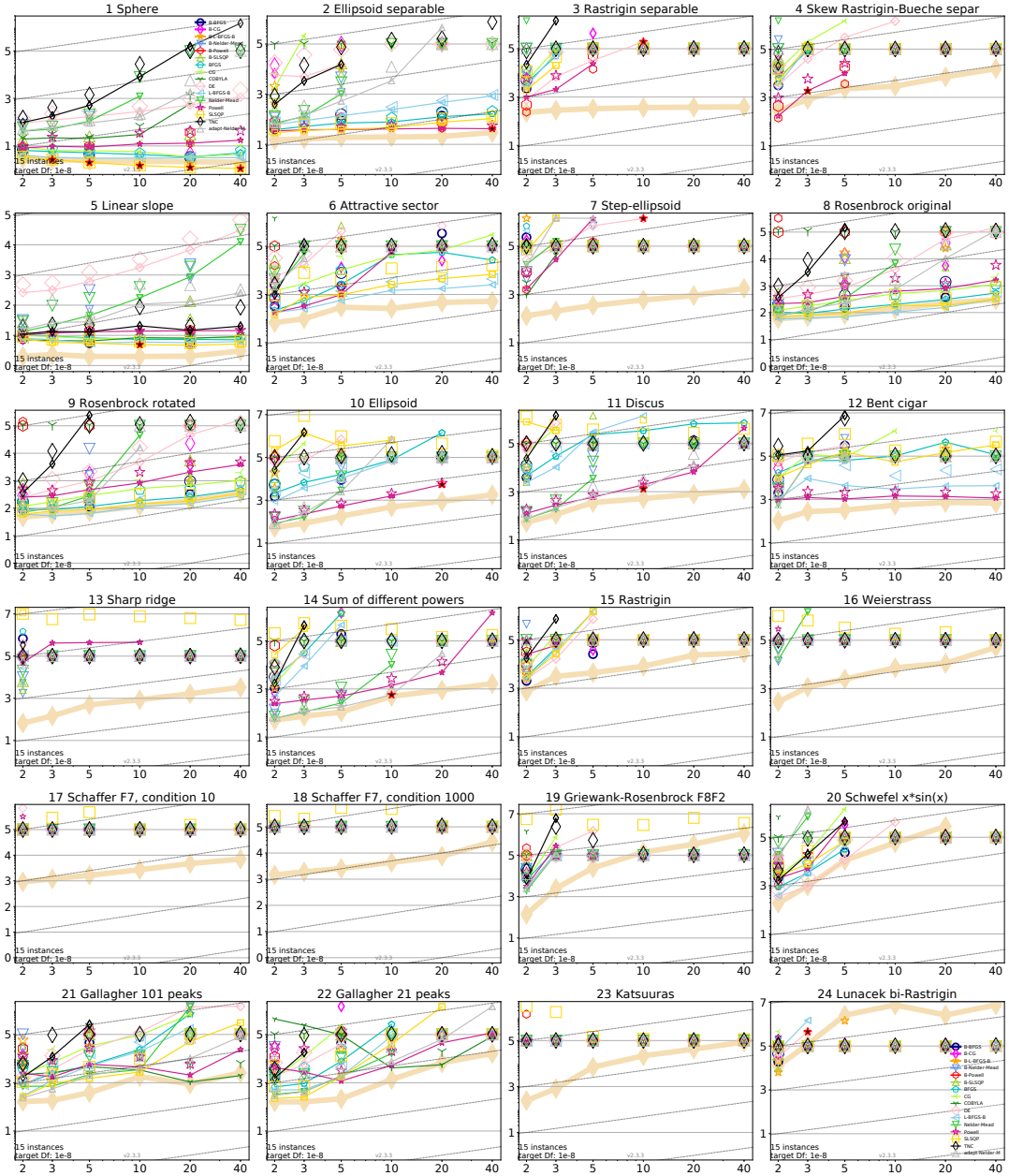

Figure 4: Average running time (aRT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: B-BFGS, ◇: B-CG, ☆: B-L-BFGS-B, ▽: B-Nelder-Mead, ⬡: B-Powell, △: B-SLSQP, ⬠: BFGS, ›: CG, ᵞ: COBYLA, ◇: DE, ◁: L-BFGS-B, ▽: Nelder-Mead, ☆: Powell, □: SLSQP, ◇: TNC, △: adapt-Nelder-Mead

and $[50D, 400D]$ for dimensions 5 and 20 respectively, while the runtimes always differ less than by a factor of 4.

For Nelder and Mead's method, adaptation of parameters is crucial. Without this option, the algorithm is deteriorated as the dimension increases. In dimension 20, the smallest target values for

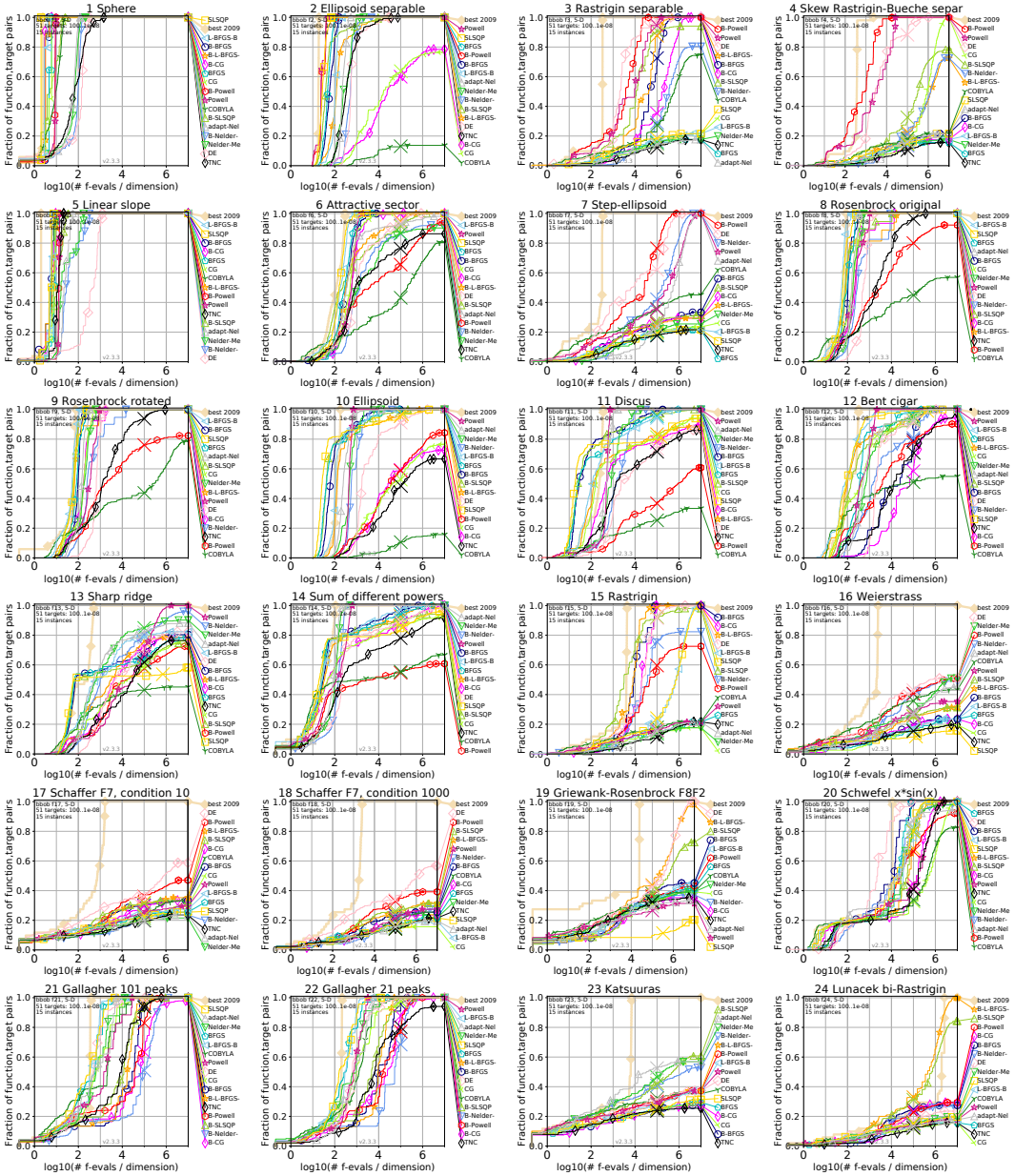

Figure 5: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 5.

the Sphere function are not reached while in the aggregated ECDF the method is dominated by all other solvers and for all budgets.

It is interesting that COBYLA, that is based simply on linear interpolation, often achieves better performance for the fraction of easiest targets than all other solvers e.g. for the Sharp Ridge and Sum of Different Powers functions in $20D$, even outperforming the virtual best solver of BBOB 2009 for small budgets. More remarkable is the performance on the multimodal Gallagher and Katsuura functions in $20D$, where it is one of the most effective methods.

Finally, Differential Evolution shows the best performance among the other (local) solvers for the group of multimodal functions with adequate global structure, where also the Basin Hopping policy is of advantage. Even though the effectiveness of DE weakens with increasing dimensionality, it maintains the highest success rate in this function group.

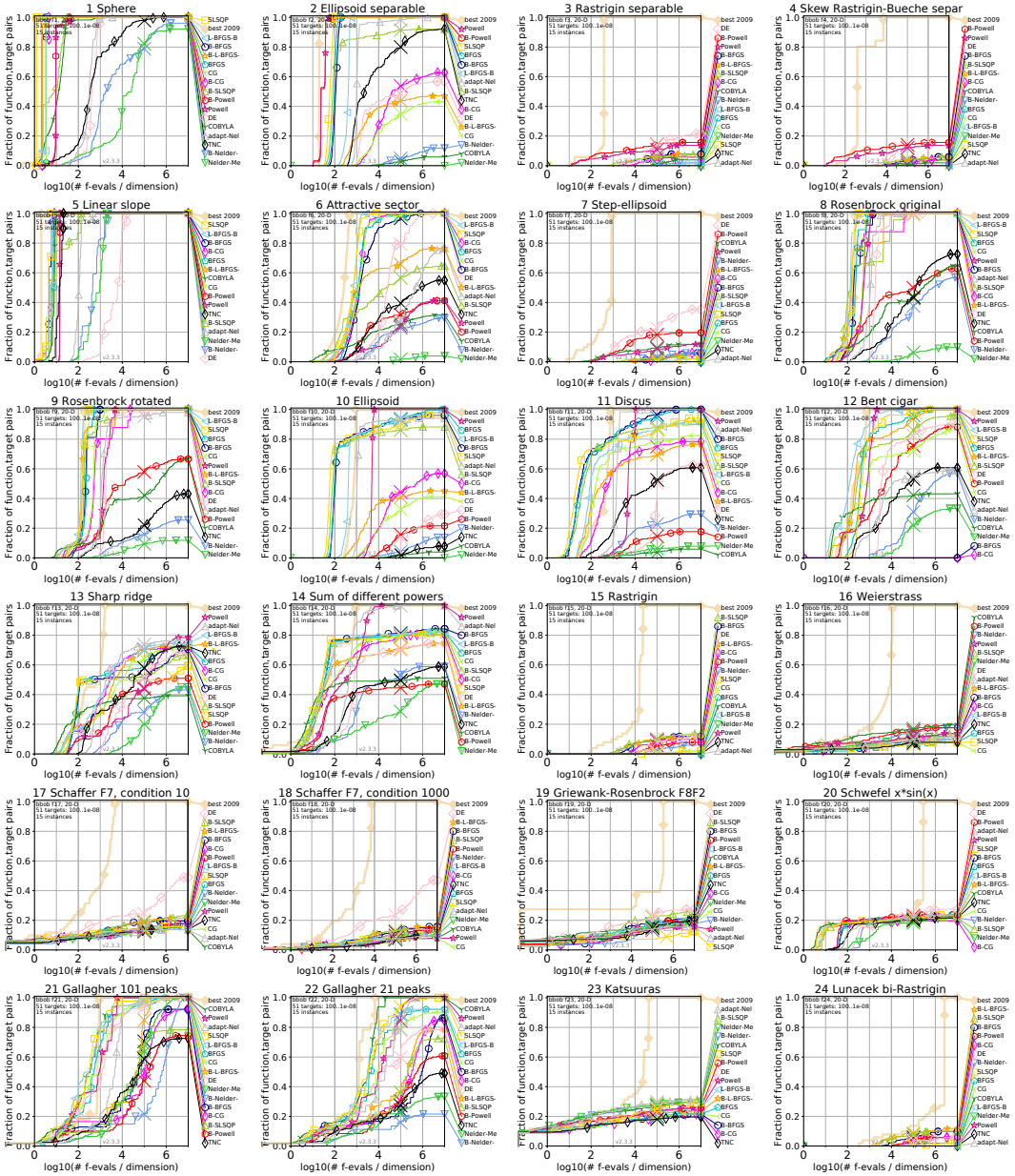

Figure 6: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 20.

7 ACKNOWLEDGEMENTS

The PhD thesis of Konstantinos Varelas is funded by the French MoD DGA/MRIS and Thales Land & Air Systems. The PhD thesis of Marie-Ange Dahito is funded by PSA Group.

REFERENCES

[1] P. Baudiš. 2014. COCOpf: An algorithm portfolio framework. *arXiv preprint arXiv:1405.3487* (2014).

[2] A. Blelly, M. Felipe-Gomes, A. Auger, and D. Brockhoff. 2018. Stopping criteria, initialization, and implementations of BFGS and their effect on the BBOB test suite. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1513–1517.

[3] S. Finck, N. Hansen, R. Ros, and A. Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Presentation of the Noiseless Functions*. Technical Report 2009/20. Research Center PPE. http://coco.lri.fr/downloads/download15.03/bbobdocfunctions.pdf Updated February 2010.

[4] F. Gao and L. Han. 2012. Implementing the Nelder-Mead simplex algorithm with adaptive parameters. *Comp. Opt. and Appl.* 51 (2012), 259–277.

[5] N. Hansen, A Auger, D. Brockhoff, D. Tušar, and T. Tušar. 2016. COCO: Performance Assessment. *ArXiv e-prints* arXiv:1605.03560 (2016).

[6] N. Hansen, A. Auger, S. Finck, and R. Ros. 2012. *Real-Parameter Black-Box Optimization Benchmarking 2012: Experimental Setup*. Technical Report. INRIA. http://coco.gforge.inria.fr/bbob2012-downloads

[7] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv*

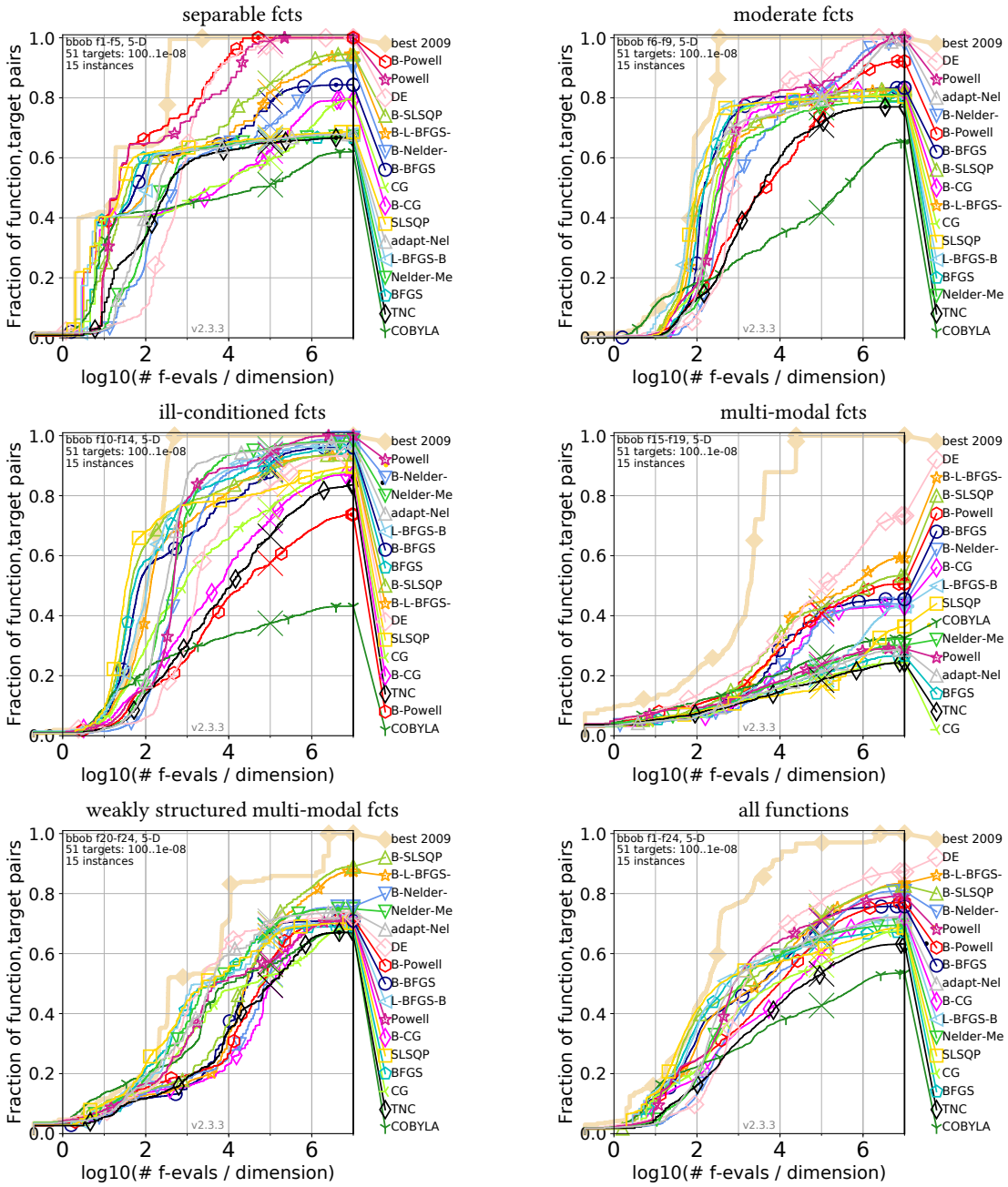

Figure 7: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

e-prints arXiv:1603.08785 (2016).

[8] N. Hansen, S. Finck, R. Ros, and A. Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions.* Technical Report RR-6829. INRIA. http://coco.lri.fr/downloads/download15.03/bbobdocfunctions.pdf Updated February 2010.

[9] N. Hansen, S. Finck, R. Ros, and A. Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions.* Technical Report RR-6829. INRIA. http://coco.gforge.inria.fr/bbob2012-downloads Updated February 2010.

[10] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. 2016. COCO: The Experimental Procedure. *ArXiv e-prints* arXiv:1603.08776 (2016).

[11] D. Kraft. *A software package for sequential quadratic programming.* Technical Report.

[12] D. C. Liu and J. Nocedal. 1989. On the Limited Memory BFGS Method for Large Scale Optimization. *Math. Program.* 45, 3 (Dec. 1989), 503–528.

[13] S. G. Nash. 1984. Newton-type minimization via the Lanczos method. *SIAM J. Numer. Anal.* 21, 4 (1984), 770–788.

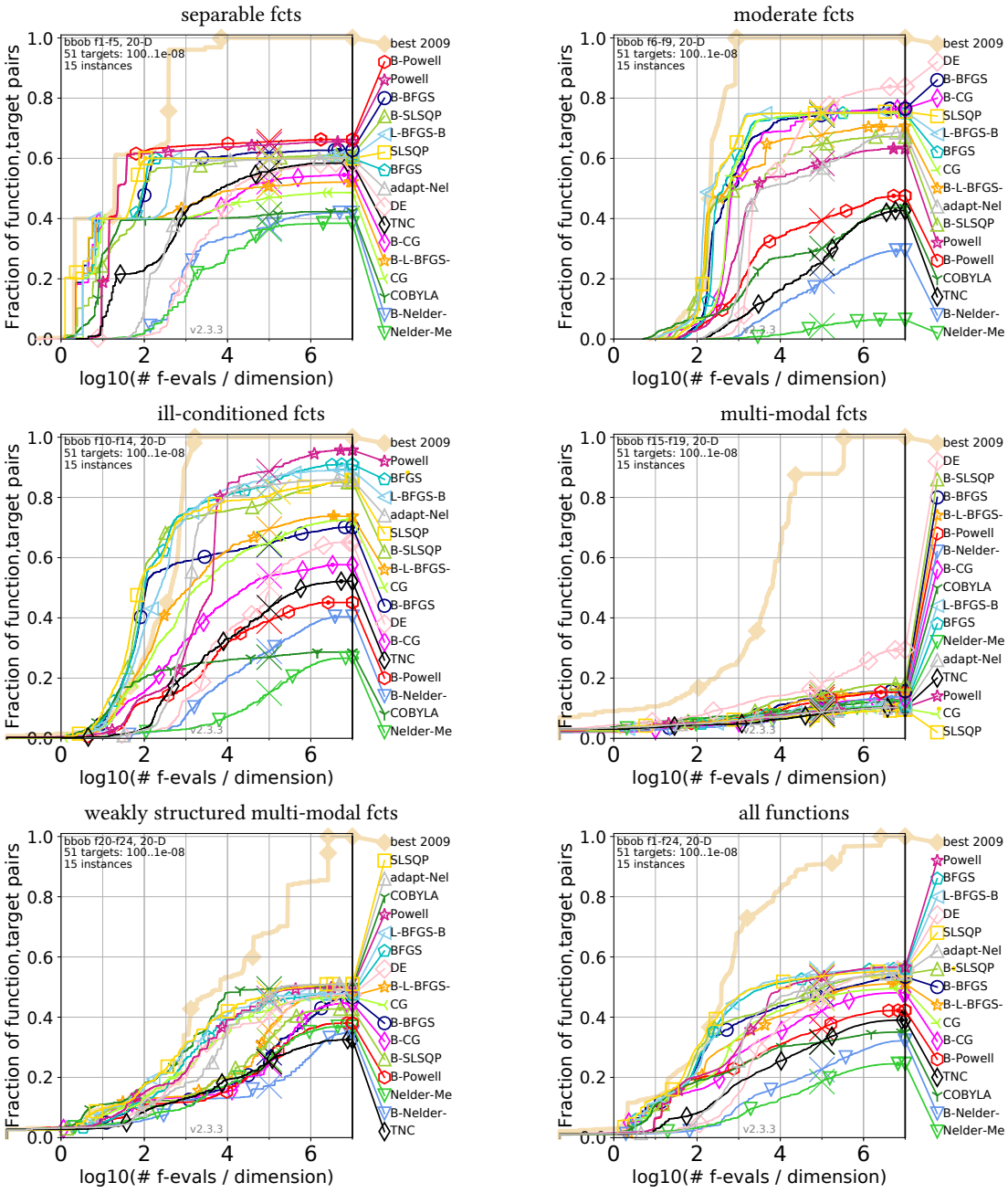

Figure 8: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 20-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

[14] J. A. Nelder and R. Mead. 1965. A simplex method for function minimization. *The computer journal* 7, 4 (1965), 308–313.

[15] J. Nocedal and S. Wright. 2006. *Numerical optimization.* Springer Science & Business Media.

[16] M. J. D. Powell. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.* 7, 2 (01 1964), 155–162. https://doi.org/10.1093/comjnl/7.2.155 arXiv:http://oup.prod.sis.lan/comjnl/article-pdf/7/2/155/959784/070155.pdf

[17] M. J. D. Powell. 1994. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*. Springer, 51–67.

[18] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press.

[19] K. Price. 1997. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*. IEEE, Piscataway, NJ, USA, 153–157. https://doi.org/10.1109/ICEC.1997.592287

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

[21] D. J. Wales and J. P. Doye. 1997. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A* 101, 28 (1997), 5111–5116.