Openly Revisiting Derivative-Free Optimization

Jeremy Rapin
Facebook AI Research

Pauline Dorval
Facebook AI Research

Jules Pondard
Facebook AI Research

Nicolas Vasilache
Facebook AI Research

Marie-Liesse Cauwet
Mines de Saint-Etienne

Camille Couprie
Facebook AI Research

Olivier Teytaud
Facebook AI Research

ABSTRACT

This paper surveys and compares a wide range of derivative-free optimization algorithms in an open source context. We also propose a genetic variant of differential evolution, an adaptation of population control for the multimodal noise-free case, new multiscale deceptive functions, and as a contribution to the debate on genetic crossovers, a test function with useless variables on which 2-points crossover achieves great performance. We include discrete and continuous and mixed settings; sequential and parallel optimization; rotated, separable and partially rotated settings; noisy and noise-free setting. We include real world applications and compare with recent optimizers which have not yet been extensively compared to the state of the art.

ACM Reference format:
Jeremy Rapin, Pauline Dorval, Jules Pondard, Nicolas Vasilache, Marie-Liesse Cauwet, Camille Couprie, and Olivier Teytaud. 2019. Openly Revisiting Derivative-Free Optimization. In *Proceedings of Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, July 13–17, 2019 (GECCO '19 Companion),* 2 pages.
https://doi.org/10.1145/3319619.3321966

We propose a platform for derivative-free optimization, filling a gap in the existing open source offer. We also provide an extensive comparison of algorithms, including algorithms which have not yet been extensively compared to the state of the art [2, 5, 10] and machine learning applications. We consider corruption of the objective function by noise, with variance not vanishing at the optimum and with a distinction between recommendation and exploration. We document the impact of the number of workers in parallel settings. We consider corruption by critical variables and/or modularity properties. We include uni/multimodal cases and ill-conditioning. Experiments can be reproduced using [20] (version 0.1.3). The command lines for reproducing our experiments are the followings for ill-conditioned optimization (resp. for the discrete setting and for our multiscale deceptive functions):

```
python -m nevergrad.benchmark illcondi –plot –num_workers 20
```

(or discrete, or deceptive, instead of illcondi)

Our benchmark includes the following families of optimization algorithms. One-shot optimization methods, which use a number of computation units equal to the budget - all candidates are evaluated simultaneously: random, Halton & Hammersley search [8, 9], latin hypercube sampling [14], and variants. Some evolution strategies [21] are evaluated as well: the simple $(1 + 1)$-algorithm implicitly parallelized by the ask/ tell/recommend interface discussed below, Covariance Matrix Adaptation, and other non-elitist (μ, λ)-methods. We include tools from mathematical programming, namely Cobyla [19], Particle Swarm Optimization

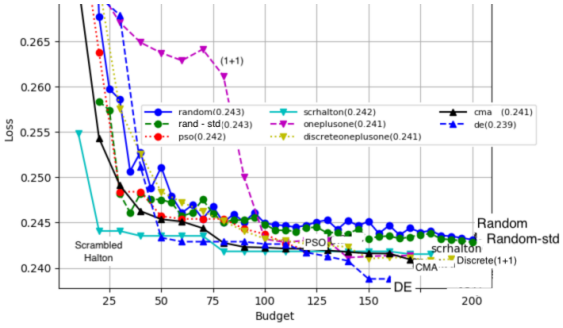

Figure 1: Comparison between various optimizers on the F_2F_5 model. Parallelism: 30 models are trained at any given time, the budget takes into account the parallelism.

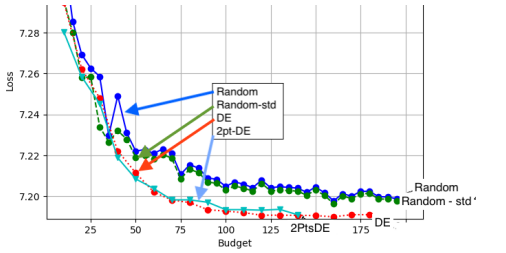

Figure 2: Comparison of various optimizers on the AIR model. Parallelism: 5 models are under training at any given time, the budget takes into account the parallelism. Variants of DE all clearly outperform random search.

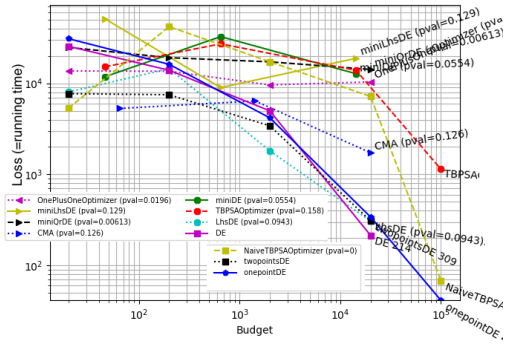

Figure 3: Parallel (84 workers) optimization of flags for JIT compilation as in Tensor Comprehension. DE variants perform well as in many of our real-world experiments.

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

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3321966

(PSO [13, 24]), bandit methods [3], Powell [18] as modified in PyOpt [17], Sequential Quadratic Programming (SQP [1]), Nelder-Mead [16]. We also include Bayesian optimization [12, 22], neural optimization as in [2] (termed NAS due to the origin in Neural Architecture Search - can be used as a standalone generic derivative-free optimization algorithm), and Differential Evolution (DE) in various flavors [23] including rotationally invariant (RotInv) or almost invariant versions [15], LHS initialization (LhsDE) and a novel use of k-point [11] crossover (CO) in lieu of the classical pointwise crossover of DE. For noise-management, we include variants of algorithms above based on repeated evaluations, and a method termed "TBPSA" (test-based population-size adaptation), based on population control (PC [10]). We also use NaiveTBPSA, which uses the individual which got the best fitness so far (BSF) as a recommendation (i.e. an approximation of the optimum) rather than the center of current Gaussian (which is traditional for PC).

Conclusion. We open source a library of test beds and derivative-free optimization algorithms. Artificial experiments can be reproduced by a single command. Contrarily to parallel, noisy and robust cases for which we get clear results, we note that the case of ill-conditioned unimodal functions appears quite complicated, depending on many factors. Results are clearly different for Cigar and for Ellipsoid. They also depend on parallelism and rotation. Covariance Matrix Adaptation (CMA) nonetheless looks like a stable tool when dimensionality is tractable. The computational cost of CMA increases much more than PSO (Particle Swarm Optimization) or DE when the dimension increases. CMA is not, contrarily to PSO and DE, invariant by addition of unimportant variables. RotInvDE and AlmostRotInvDE perform excellently when the parallelism exceeds the dimension [15].For real world hyperparameter search, DE significantly outperforms RS in a wide range of dimensions (from 2 to 26), including high parallelism with just 3 or 4 generations. It also outperformed PSO and CMA in Fig. 1. Consistently with [15], DE with almost rotational invariance (i.e. $CR = 0.9$) performs better than other DE when the test is fully rotated. Nonetheless, it performed poorly in some real-world tasks. DE was surprisingly robust for the tuning of machine learning tasks on real data. The versions of DE with genetic CO were good as well, without clear improvement compared to variable-wise CO in some cases, but also with huge improvement in some clear simple illustrative test functions, compared to many state-of-the-art optimization methods: this is an element in the long standing debate on the effectiveness of genetic operators [6, 7, 11]. We provide new deceptive functions, illustrating the robustness of PSO. These functions are complementary to the known jump function and the functions with critical variables discussed in Rapin and Teytaud [20]. We got excellent results for mathematical programming techniques (gradient free) in the case of very smooth functions such as the Cigar function. Neural optimization [2] is compared to classical existing algorithms and to PC methods from [10] in noisy, multimodal and simple cases. High dimension is hard for neural optimization. PC is excellent for noise in continuous domains. Neural optimization can be competitive for noisy optimization and/or hard multimodal optimization when dimension is moderate and budget is large - and there might be a significant headroom for these recent methods. Population control [10] works quite well in noise-free multimodal optimization - *when using the BSF recommendation* policy. In the discrete world, we confirm good results of FastGA [5] and of the uniform mixing of mutation rates [4].

REFERENCES

[1] SME Artelys. 2015. (2015). https://www.artelys.com/news/159/16/KNITRO-wins-the-GECCO-2015-Black-Box-Optimization-Competition

[2] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. 2017. Neural Optimizer Search with Reinforcement Learning. In Proc. of the 34th International Conference on Machine Learning (ICML'17), Vol. 70. 459–468.

[3] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. 2011. Pure exploration in finitely-armed and continuous-armed bandits. Theor. Comput. Sci. 412, 19 (2011), 1832–1852.

[4] Duc-Cuong Dang and Per Kristian Lehre. 2016. Self-adaptation of Mutation Rates in Non-elitist Populations. In Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings. 803–813.

[5] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast Genetic Algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17). ACM, New York, NY, USA, 777–784.

[6] Carola Doerr and Eduardo Carvalho Pinto. 2018. A Simple Proof for the Usefulness of Crossover in Black-Box Optimization. In PPSN 2018: Parallel Problem Solving from Nature – PPSN XV (Lecture Notes in Computer Science), Vol. 11102. Springer, Coimbra, Portugal, 29–41.

[7] David B. Fogel and Lauren C. Stayton. 1994. On the effectiveness of crossover in simulated evolutionary optimization. Biosystems 32, 3 (1994), 171 – 182.

[8] J.H. Halton. 1960. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer. Math.* 2 (1960), 84–90.

[9] J. M. Hammersley. 1960. Monte-Carlo Methods For Solving Multivariate Problems. Annals of the New York Academy of Sciences 86, 3 (1960), 844–874.

[10] Michael Hellwig and Hans-Georg Beyer. 2016. Evolution Under Strong Noise: A Self-Adaptive Evolution Strategy Can Reach the Lower Performance Bound - The pcCMSA-ES. In Parallel Problem Solving from Nature – PPSN XIV, Julia Handl, Emma Hart, Peter R. Lewis, Manuel López-Ibáñez, Gabriela Ochoa, and Ben Paechter (Eds.). Springer International Publishing, Cham, 26–36.

[11] John H. Holland. 1973. Genetic Algorithms and the Optimal Allocation of Trials. SIAM J. Comput. 2, 2 (1973), 88–105.

[12] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization 13, 4 (01 Dec 1998), 455–492.

[13] James Kennedy and Russell C. Eberhart. 1995. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks. 1942–1948.

[14] M. D. McKay, R. J. Beckman, and W. J. Conover. 1979. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics 21, 2 (1979), 239–245.

[15] J. Montgomery and S. Chen. 2010. An analysis of the operation of differential evolution at high and low crossover rates. In IEEE Congress on Evolutionary Computation. 1–8.

[16] John A. Nelder and Roger Mead. 1965. A simplex method for function minimization. Computer Journal 7 (1965), 308–313.

[17] Ruben E. Perez, Peter W. Jansen, and Joaquim R. R. A. Martins. 2012. pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization. Structures and Multidisciplinary Optimization 45, 1 (2012), 101–118.

[18] 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 (1964), 155–162.

[19] M. J. D. Powell. 1994. A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. Springer Netherlands, Dordrecht, 51–67.

[20] J. Rapin and O. Teytaud. 2018. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad. (2018).

[21] I. Rechenberg. 1973. Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution. Fromman-Holzboog Verlag, Stuttgart.

[22] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In Advances in Neural Information Processing Systems 25 (NIPS'12). 2951–2959.

[23] Rainer Storn and Kenneth Price. 1997. Differential Evolution: A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. J. of Global Optimization 11, 4 (Dec. 1997), 341–359.

[24] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas. 2013. Standard Particle Swarm Optimisation 2011 at CEC-2013: A baseline for future PSO improvements. In 2013 IEEE Congress on Evolutionary Computation. 2337–2344.