Optimizing with Low Budgets: a Comparison on the Black-box Optimization Benchmarking Suite and OpenAI Gym – SUPPLEMENTARY MATERIAL

Elena Raponi, Nathanaël Carraz Rakotonirina, Jérémy Rapin, Carola Doerr, Olivier Teytaud

***Abstract*—In this supplementary document, we delve into the key distinctions between the BBOB benchmarking suite in the COCO environment and Nevergrad's YABBOB. We also outline the process for replicating our results for BBOB and conducting experiments on OpenAI Gym using Nevergrad. Additionally, we elaborate on the results obtained for larger neural networks and increased evaluation budgets. Finally, we furnish a comprehensive list of all the methods compared in our study.**

I. Discussion: differences between BBOB and Nevergrad's YABBOB

There are differences between the benchmarking suites.

A. Domains

First, working in unbounded domains with a Gaussian distributed optimum (as in some benchmarks in Nevergrad) leads to differences compared to benchmarks on bounded domains such as BBOB, especially for functions with an optimum at the boundary. Both contexts look interesting, but some adaptation of NGOpt was needed to make it tackle optima on the boundary. In addition, we add bounded counterparts of YABBOB termed YABOUNDEDBBOB and YABOXBBOB for facilitating further research with simultaneously the convenience of Nevergrad (see Section I-C) and the bounded setting of BBOB.

B. Budgets and independence

In BBOB, when specifying a maximum budget of $100D$, the results plotted for a lower budget of $10D$ are obtained as a truncation of the $100D$-budget runs. On the contrary, Nevergrad runs each budget separately, which is more expensive from a computational point of view but gives a more complete picture of the different budgets. This can be remedied by launching distinct runs for different budgets for BBOB.

C. Parallelization

Nevergrad was more suitable than BBOB for testing very slow algorithms like AX [1] because it is easy to massively parallelize it on a cluster.

E. Raponi (the corresponding author) is with LIACS, Leiden University, Leiden, The Netherlands (e-mail: e.raponi@liacs.leidenuniv.nl). N. C. Rakotonirina is with Universitat Pompeu Fabra, Barcelona, Spain (e-mail: nathanael.rakotonirina@upf.edu). J. Rapin and O. Teytaud are with Meta AI Research, Paris, France (e-mail: jrapin, oteytaud@fb.com). C. Doerr is with LIP6, CNRS, Sorbonne Université, Paris, France (e-mail: carola.doerr@lip6.fr).

II. BBOB interfaces

A strength of BBOB is that the interfacing is quite easy, greatly facilitating reproducibility [2].

```
% Nevergrad's NGOpt.
% Also SMAC, SMAC2, AX, BO: using Nevergrad's API.
ng.optimizers.NGOpt(ng.p.Array(lower=lbounds, upper=ubounds, shape=[dim]), num_workers=1, budget=evals).minimize(f)

% HyperOpt.
fmin(fn=lambda x: f([x['w'+str(i)] for i in range(dim)]), space={'w'+str(i): hp.uniform('w' + str(i), -5, 5) for i in range(dim)}, algo=tpe.suggest, max_evals=evals)

% Optuna.
class OptunaObjective(object):
    def __init__(self, problem):
        self.problem = problem

    def __call__(self, trial):
        x = []
        for i in range(self.problem.dimension):
            x.append(trial.suggest_float("x{}".format(i), problem.lower_bounds[i], problem.upper_bounds[i]))
        return self.problem(x)

study = optuna.create_study(direction="minimize")
study.optimize(OptunaObjective(problem), n_trials=evalsleft())

% Turbo.
class turbo_function:
    def __init__(self, dim=len(lbounds)):
        self.dim = dim
        self.lb = lbounds
        self.ub = ubounds

    def __call__(self, x):
        assert len(x) == self.dim
        assert x.ndim == 1
        assert np.all(x <= self.ub) and np.all(x >= self.lb)
        return f(x)

my_turbo = Turbo1(f=turbo_function(), lb=lbounds, ub=ubounds, max_evals=evals, n_init=min(evals, 20))

my_turbo.optimize()

% LA-MCTS.
% We tested several successive variants of the code,
% without much impact: the version below is the last.
% We also tested several values
% of ninits, without much change.
```

```
agent = MCTS(lb = f.lb,
             ub = f.ub,
             dims = f.dims,
             ninits = 40,  # We tested variants without much change.
             func = f
             )
agent.search(iterations = evalsleft())
```

III. How to run the experiments on OpenAI Gym

OpenAI Gym was recently introduced in Nevergrad. After cloning Nevergrad at https://github.com/facebookresearch/nevergrad.git, experiments can be launched with the following command line:

```
python -m nevergrad.benchmark ng_full_gym --repetitions=10 --plot
```

However, in order to run a reduced experiment like the one presented in this paper, we changed the definition of the budget, dimension, and scaling (budget $25, 50, 100, 200, 400, 800$, scaling-factor 1, and add a limit 40 to the dimension) in the ng_full_gym experiment in nevergrad/benchmarks/gymexperiments.py (Line 80) for obtaining the setup as in Section III of the main paper.

IV. Bigger neural nets for OpenAI Gym

Here, we offer additional details about the configuration employed to produce the results depicted in Fig. 5 in the main manuscript. Bigger neural networks provide a more complete version of the Ng-Full-Gym problem, where the neural factor parameter, which scales the size of the neural networks, is equal to 3. This does not change the optimization methods, only the scale of the problems towards a greater dimension. Dimensions up to 264 are considered. This benchmark is unbounded, the the scale of the algorithms (i.e., the standard deviation of the first samples) is therefore particularly critical and makes a fair comparison difficult.

We note that PSO and HyperOpt are still the best or among the best for each considered budget. BO-based methods (BO and AX) perform well for limited budgets and algorithms specific for RL tasks (NGOpt16RL and SpecialRL) become always more competitive as the budget increases.

V. Budgets 800, 1600, and 3200

While Figs. 4 and 5 in the main paper present results restricted to budgets of at most 400 function evaluations, Fig. 1 presents results for larger budgets of 800, 1600, and 3200 evaluations applied to multi-deterministic Open AI Gym with both tiny and (for methods which are computationally cheap enough) bigger neural nets. As in [3], CMA or NGOpt get better as the budget grows. We also find that, while most BO-based methods weaken, HyperOpt performs satisfactorily.

VI. List of methods

Table I contains all algorithms involved in our comparison. They are listed in alphabetical order. For each method, we give the labels used in the plots, the testbeds on which they were tested (BBOB, OpenAI Gym, or both), and a brief description of the algorithm.

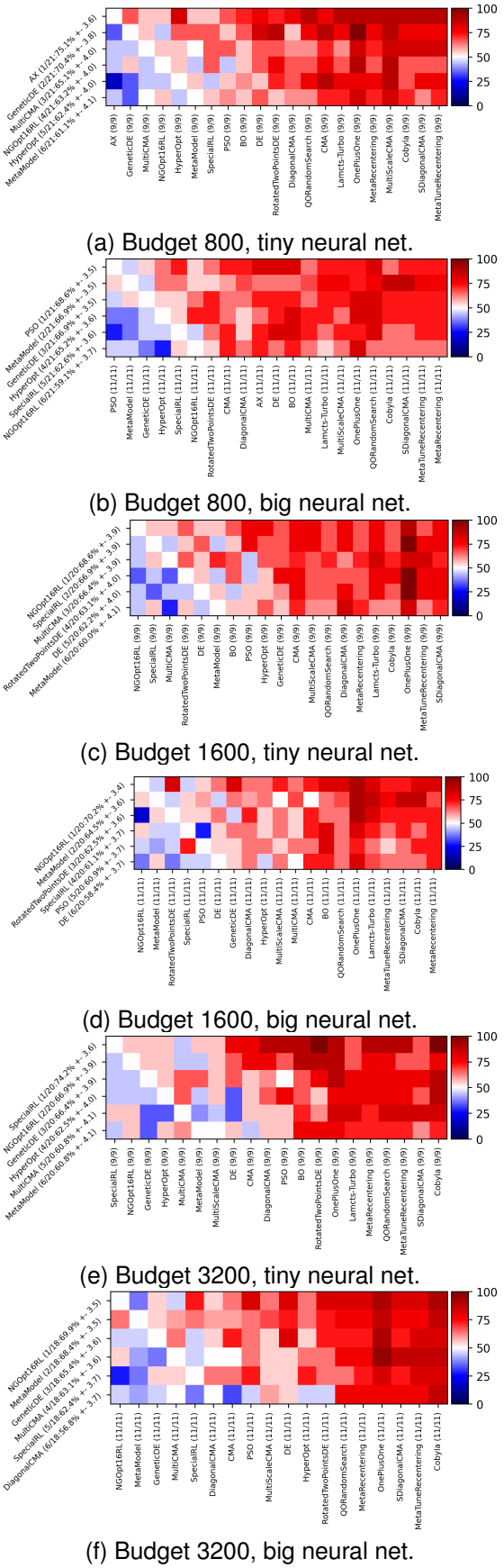

Fig. 1. Extension of Figs. 4 and 5 in the main manuscript, for budget 800, 1600, and 3200, for algorithms that were sufficiently fast (faster than 3 days of wall-clock time).

TABLE I
ALGORITHMS INVOLVED IN OUR COMPARISON. OAIG = OPENAI GYM

Name	Label	Testbed	Description
AX [1]	ax, AX	BBOB, OAIG	A modular BO framework that uses BoTorch primitives for optimization over continuous spaces. It automates the selection of optimization routines, reducing the amount of fine-tuning required.
BO [4]	BO	BBOB, OAIG	The Bayesian Optimization algorithm [4] implemented in Nevergrad. The python class is a wrapper over the bayes_opt package [5].
CMA-ES [6]	cmafmin2, CMA	BBOB, OAIG	Covariance Matrix Adaptation Evolution Strategy, a state-of-the-art optimizer in evolutionary computation for non-linear non-convex BBO problems.
Cobyla [7]	Cobyla	BBOB, OAIG	Constrained Optimization By Linear Approximation optimizer. A sequential trust-region algorithm for derivative-free constrained problems. It employs linear approximations to the objective and constraint functions.
DefaultCMA [6]	defaultcma	BBOB	A version of CMA without the BBOB-specific initialization used for the experiments of CMA on BBOB.
Diagonal CMA-ES [6]	DiagonalCMA	OAIG	Version of CMA-ES made faster by a diagonal covariance matrix.
Differential Evolution [8]	DE	OAIG	Classical differential evolution algorithm.
Genetic DE [9]	GeneticDE	OAIG	Hybrid algorithm that performs 200 iterations with RotatedTwoPointsDE and then uses TwoPointsDE.
HyperOpt [10]	hyperopt, HyperOpt	BBOB, OAIG	Library for serial and parallel hyperparameter optimization, designed to accommodate Bayesian optimization algorithms based on Gaussian processes and regression trees. We use the version based on Parzen estimates.
LA-MCTS [11]	lamcts5, Lamcts-Turbo	BBOB, OAIG	MCTS-based derivative-free meta-solver that recursively learns a space partition in a hierarchical manner. Sampling and minimization are then performed in the selected region using TuRBO-1 (no bandit).
MetaModel [9]	MetaModel	OAIG	Nevergrad's solver that adds a metamodel to an optimizer, which is defined as an input parameter. Unless specified otherwise, the base algorithm is CMA-ES.
MetaRecentering [12]	MetaRecentering	OAIG	One-shot, fully parallel optimization method using a Hammersley sampling scheme. A variant of Random Search that ensures a higher uniformity.
MetaTuneRecentering [13]	MetaTuneRecentering	OAIG	Similar to MetaRecentering but with a more sophisticated parametrization. The population is sampled according to a normal distribution with a variance calibrated to the population's size and the problem's dimension.
MultiCMA [9]	MultiCMA	OAIG	Nevergrad's solver that splits the search budget in two. In the first portion, it tries different search algorithms; then it uses the rest of the budget on the strategy that worked the best. Here, the compared algorithms are three different initializations of CMA-ES.
MultiScale CMA-ES [9]	MultiScaleCMA	OAIG	Same algorithm as MultiCMA, but the CMA-ES versions have different scales.
NGOpt [9]	ngopt16	BBOB	A wizard that combines many classical algorithms in various ways based on the problem definition, without human intervention [3]. For sequential, low-dimensional, and noise-free problems, it mainly uses CMA, Cobyla, and (1+1)-type sampling equipped with metamodels.
NGOptRL [9]	NGOpt16RL	OAIG	NGOpt version optimized for reinforcement learning tasks. It uses a specific bit of information, as it uses the knowledge that the problem is an RL problem. It is a bet-and-run of DiagonalCMA, PSO and GeneticDE.
OnePlusOne [14]	OnePlusOne	OAIG	(1+1) evolutionary algorithm with the one-fifth adaptation rule.
Optuna [15]	optuna	BBOB	Automatic hyperparameter optimization software framework which uses state-of-the-art algorithms for sampling hyperparameters and pruning unpromising trials. By default, Optuna implements a BO algorithm (Tree-structured Parzen Estimator).
Particle Swarm Optimization [16]	PSO	BBOB, OAIG	An algorithm based on moving a population of particles in the search space according to their current positions and velocities.
Quasi Opposite Random Search (inspired by [17])	QORandomSearch	OAIG	Random Search algorithm that symmetrizes exploration with respect to the center but multiplying for a random factor. "Quasi" means that the exploration is not exactly symmetric.
Rotated Two Points DE [18]	RotatedTwoPointsDE	OAIG	Based on TwoPointsDE, which is DE combined with a 2-points crossover. This version can copy and paste from one part of the list of variables to another part: the crossover is not necessarily in place.
Scaled Diagonal CMA-ES [9]	SDiagonalCMA	OAIG	Scaled version of DiagonalCMA where the initialization is divided by 1000, i.e., all coordinates are divided by 1000 (if the center is at zero, otherwise translation is applied).
SMAC [19]	SMAC	BBOB	A sequential model-based algorithm for the hyperparameter optimization of ML algorithms, specifically suitable for high dimensions and discrete input dimensions.
SMAC-HPO [19]	SMAC2	BBOB	SMAC with Hyperparameter Optimization.
SpecialRL [9]	SpecialRL	OAIG	Algorithm that runs NGOpt16RL for half the budget, and then uses test-based population size adaptation (TBPSA), an algorithm dedicated to noisy optimization [21].
TuRBO [20]	turbo	BBOB	A trust-region-inspired algorithm using Thompson sampling rather than the optimization of an acquisition function to find new candidate solutions in each subregion.
TuRBO-20 [20]	turbo20	BBOB	The multi-trust-regions counterpart of Turbo.

REFERENCES

[1] FacebookResearch, "Ax - adaptive experimentation," ax.dev, 2020.

[2] M. López-Ibáñez, J. Branke, and L. Paquete, "Reproducibility in evolutionary computation," *ACM Trans. Evol. Learn. Optim.*, vol. 1, no. 4, pp. 14:1–14:21, 2021. [Online]. Available: https://doi.org/10.1145/3466624

[3] L. Meunier, H. Rakotoarison, P. Wong, B. Rozière, J. Rapin, O. Teytaud, A. Moreau, and C. Doerr, "Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking," *IEEE Trans. Evol. Comput.*, vol. 26, no. 3, pp. 490–500, 2022. [Online]. Available: https://doi.org/10.1109/TEVC.2021.3108185

[4] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 2951–2959.

[5] F. Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python," 2014. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[6] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 11, no. 1, 2003.

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

[8] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341-–359. 1997. https://doi.org/10.1023/A:1008202821328

[9] J. Rapin and O. Teytaud, "Nevergrad - A gradient-free optimization platform," https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[10] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: a python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, no. 1, p. 014008, 2015.

[11] L. Wang, R. Fonseca, and Y. Tian, "Learning search space partition for black-box optimization using monte carlo tree search," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/e2ce14e81dba66dbff9cbc35ecfdb704-Abstract.html

[12] M. Cauwet, C. Couprie, J. Dehos, P. Luc, J. Rapin, M. Rivière, F. Teytaud, O. Teytaud, and N. Usunier, "Fully parallel hyperparameter search: Reshaped space-filling," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, vol. 119. PMLR, 2020, pp. 1338–1348. [Online]. Available: http://proceedings.mlr.press/v119/cauwet20a.html

[13] L. Meunier, C. Doerr, J. Rapin, and O. Teytaud, "Variance reduction for better sampling in continuous domains," in *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 12269. Springer, 2020, pp. 154–168.

[14] M. A. Schumer and K. Steiglitz, "Adaptive Step Size Random Search," *IEEE Trans. Automat.*, vol. AC-13, no. 3, pp. 270–276, 1968.

[15] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019*. ACM, 2019, pp. 2623–2631. [Online]. Available: https://doi.org/10.1145/3292500.3330701

[16] J. Kennedy, R. Eberhart, "Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks*. IEEE, 1995, pp. 1942–1948. [Online]. Available: doi:10.1109/ICNN.1995.488968

[17] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Quasi-oppositional differential evolution," in *2007 IEEE Congress on Evolutionary Computation*, Sep. 2007, pp. 2229–2236.

[18] J. H. Holland, "Adaptation in Natural and Artificial Systems," University of Michigan Press, 1975.

[19] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration." in *LION*, ser. Lecture Notes in Computer Science, C. A. C. Coello, Ed., vol. 6683. Springer, 2011, pp. 507–523. [Online]. Available: http://dblp.uni-trier.de/db/conf/lion/lion2011.html#HutterHL11

[20] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable global optimization via local Bayesian optimization," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/6c990b7aca7bc7058f5e98ea909e924b-Paper.pdf

[21] V. Khalidov, M. Oquab, J. Rapin, and O. Teytaud, "Consistent population control: generate plenty of points, but with a bit of resampling," Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, FOGA 2019, 116–123, Potsdam, Germany, August 27-29, 2019. Available: https://doi.org/10.1145/3299904.3340312