Parallelized Bayesian Optimization for Expensive Robot Controller Evolution

Margarita Rebolledo[1], Frederik Rehbach[1], A.E. Eiben[2], and Thomas Bartz-Beielstein[1]

[1] Institute For Data Science, Engineering, and Analytics, TH Köln, Germany
firstname.lastname@th-koeln.de
[2] Department of Computer Science, Vrije Universiteit Amsterdam, Netherlands
a.e.eiben@vu.nl

Abstract. An important class of black-box optimization problems relies on using simulations to assess the quality of a given candidate solution. Solving such problems can be computationally expensive because each simulation is very time-consuming. We present an approach to mitigate this problem by distinguishing two factors of computational cost: the number of trials and the time needed to execute the trials. Our approach tries to keep down the number of trials by using Bayesian optimization (BO) –known to be sample efficient– and reducing wall-clock times by parallel execution of trials. We compare the performance of four parallelization methods and two model-free alternatives. Each method is evaluated on all 24 objective functions of the Black-Box-Optimization-Benchmarking (BBOB) test suite in their five, ten, and 20-dimensional versions. Additionally, their performance is investigated on six test cases in robot learning. The results show that parallelized BO outperforms the state-of-the-art CMA-ES on the BBOB test functions, especially for higher dimensions. On the robot learning tasks, the differences are less clear, but the data do support parallelized BO as the 'best guess', winning on some cases and never losing.

Keywords: Parallelization · Bayesian Optimization · CMA-ES · BBOB Benchmarking · Robotics.

1 Introduction

Many real-world optimization problems are expensive to evaluate and require a considerable amount of computation time for each candidate solution evaluation. In such cases, the total evaluation budget is usually severely limited. Bayesian optimization (BO) [22,9] and parallel computing are two state-of-the-art methods for solving budget limited black-box optimization problems.

In Bayesian optimization, a data-driven surrogate of the expensive function is fitted and an extensive search on the cheap surrogate is feasible. Only the points that are considered promising on the surrogate are evaluated on the expensive function. This makes BO very sample efficient compared to other algorithms.

Parallel computing makes use of the ever-increasing amount of available CPU cores in modern server systems. Running multiple simulations in parallel requires more computational resources and more energy, yet, it does not increase the real-time spent on the computation. Population-based algorithms like evolutionary algorithms (EAs), which can propose multiple candidate solutions per iteration, have an inherent efficiency benefit on parallel computing systems. In this area the covariance matrix adaptation evolution strategy (CMA-ES) [13] is the state-of-the-art EA for real valued optimization. Other than CMA-ES, the standard BO approach does only evaluate a single candidate solution per iteration. Yet, several approaches have been proposed through which BO can be adapted to a parallel environment. An overview of these methods is given in [11].

One example of expensive, yet parallelizable, objective functions can be found in the field of evolutionary robotics (ER). ER aims to automatically design robots that are well suited to their environment and can perform a given task [4,7]. This can be achieved by evolving the robot morphologies (bodies) and controllers (brains) through iterated cycles of selection and reproduction. As outlined in the triangle of life (ToL) framework proposed by [6], this implies that newborn robots (with a new body form that is different from the bodies of the parents) must start their lifetime with a learning stage. In this stage, a robot with the given morphology needs to learn to control its body and maximize its task performance.

In this paper we investigate six robot learning test cases, specified by two different tasks, gait learning (moving in any direction) and directed locomotion, and three bio-inspired robot shapes, a snake, a spider, and a gecko. Testing the behavior of these robots requires a computationally expensive simulation. In future stages, learning will be conducted in the real world where 3D-printed robots are automatically configured with the proposed controllers and their behavior is tested in a robot arena. Each of these trials will require a considerable amount of time, ranging from multiple minutes up to an hour. This makes the real world evaluation function even more expensive than the simulator we are currently using. Due to the expensive nature of the problem at hand, extensive tests and algorithm comparisons are less feasible. Therefore, additional to the robots simulations, the well-known set of Black-Box-Optimization-Benchmarking (BBOB) test functions [17] are used for extensively comparing the performance of each algorithm on different problem classes and dimensionalities. Given the results on the artificial functions, comparisons can be drawn about real-world applications. Considering this approach the following research questions arise:

RQ-1 Can parallel variants of BO outperform inherently parallel algorithms like CMA-ES if the evaluation budget is severely constrained?
RQ-2 Which parallel variants of BO show the best performance on which of the tested problem landscapes?
RQ-3 How do BO and CMA-ES compare on the robot application?

The rest of this paper is structured as follows: We will first give an overview of the implemented parallelization methods in Section 2. We will introduce the robot application for the real-world scenario in Section 3. The setup of our

experiments is explained in Section 4. The obtained results are presented in Section 5 and finally discussed in Section 6.

2 Overview of implemented optimization methods

2.1 Bayesian Optimization

Bayesian optimization (BO) is an iterative global optimization framework useful for expensive black-box derivative free problems [22,9]. The main components of BO are a surrogate model and an acquisition function. We use Gaussian process(GP) [19] with radial basis function kernel [8] as the surrogate model. The kernel is defined as $\Sigma^{(i)} = \exp(-\sum_{j=1}^{n} \theta_j |x_j^{(i)} - x'_j|^{p_j})$, where p_j determines the correlation function smoothness and θ_j the extend of a point's x_i influence. Two variants of this kernel will be used in this work: **P2**, where $p_i = 2$, and **FitP**, where p_i is part of the optimization loop.

Three of the most common acquisition functions are considered: expected improvement (EI), lower confidence bound (LCB) and predicted value (PV) [22]. PV is a heavy exploitation approach. It uses the surrogate model's best prediction as the next candidate solution. Under the right conditions this assures quick convergence but might lead to getting stuck in local optima.

LCB is an optimistic acquisition function. At a point x, it underestimates the mean using the uncertainty, $\alpha_{LCB}(x, \beta)$, where $\beta \leq 0$ is the exploration-exploitation trade-off factor. Following [23] we set $\beta = 1$ for every instance of BO using LCB as acquisition function.

EI [8] is a more explorative acquisition function. It calculates the amount of improvement a new point can achieve based on its mean and variance. The point with the highest expected improvement is selected as the next candidate solution.

2.2 Parallelization Approaches

Depending on the selected acquisition function BO can balance model exploitation (to quickly converge to the global optimum) and model exploration (to increase model quality). Most parallelization techniques for BO create multiple candidates per iteration by searching for multiple differently weighted compromises of these two goals. A total of q candidate solutions is generated per iteration, where q defines the number of objective function evaluations that can be run in parallel. In the following, three of these techniques are presented: investment portfolio improvement (IPI), multi-point expected improvement (q-EI), multi-objective infill criteria (MOI), and our implementation, multi-kernel Bayesian optimization (mK-BO). For more information about these methods please refer to the presented bibliography.

Investment Portfolio Improvement [24] views the suggestion of new candidate solutions as handling an investment portfolio. It tries to balance high- and low-risk investments. On the one hand, candidate solutions which have a very high probability of improvement are safe investments. Yet, they often yield a minimal improvement over the best-known candidate solutions. On the other hand, candidates with a high expected improvement usually incorporate a high uncertainty and thus high risk. [24] proposes a sequential switching criterion that cycles between a high-, medium,- and a low-risk point. This sequential approach can directly be adapted to a parallel application. Instead of just three candidates, a different balance between exploration and exploitation can be defined for q candidate solutions. The candidates are then evaluated in parallel.

Multi-point Expected Improvement in q-EI, the definition of EI is adapted to a set of points with shared EI. A detailed description of an efficient implementation is given by Ginsbourger et al. [10]. For this study, the q-EI implementation of the 'DiceOptim' R-package [21] is used together with the model implementation of the 'DiceKriging' package [21].

A property of q-EI that is worth mentioning is that it favors solutions that are spread throughout the search space. As sequential EI already leans towards exploration, this effect is fortified with each added parallel candidate. If two points of a set are too close to each other, the expected improvement of one of them will tend to zero.

Multi-objective Infill Criteria [2] approaches the compromise between exploration and exploitation as a multi-objective optimization problem. The predicted value of the GP model defines the first objective. The models' uncertainty defines the second. An evolutionary algorithm searches for the Pareto-front of the bi-objective acquisition function. Each point on the front is a good compromise between the objectives and could be considered in the parallel evaluation. To narrow down the number of proposed points Bischl et al. consider two distance-based techniques (nearest neighbor and nearest better neighbor) as a third objective on the Pareto-front. For our experiments, we chose the implementation that was considered best in their benchmarks, described in [2] with ID 10. Our implementation is directly taken from the author's R-package 'mlrMBO' [1].

Multi-Kernel BO additionally, we inspect a rather simple way to parallelize BO. When faced with a black-box optimization problem, often the choice for the right kernel parameter setting or acquisition function for BO is not clear. While some settings might perform well on specific landscape types or problem dimensionalities, they might fail in others. In a parallel environment, this problem can be circumvented by running different BO configurations in parallel. Instead of choosing a single setup for BO, as many configurations are created as objective function evaluations can be run in parallel. Since our parallelization system is capable of running six robot simulations in parallel, we describe six

distinct BO configurations to be run in parallel: Three acquisition functions, EI, LCB, and PV, are combined with the previously mentioned variants of kernel configuration, P2 and FitP.

After sampling an initial design, the algorithm will build the six distinctly configured GP models in parallel, one on each available core. Their respective acquisition functions are optimized, and the candidate solutions are evaluated. The algorithm waits for all instances to complete their evaluations in a synchronization step. After that, the next iteration starts with a new set of models built on all so far evaluated candidate solutions. Thus, the models share knowledge and synchronize after each iteration.

2.3 CMA-ES

CMA-ES is a state-of-the-art EA for optimizing non-linear non-convex black-box functions. In short, CMA-ES samples in each iteration a population , λ, from a multivariate normal distribution $\mathcal{N} \sim (m, \sigma^2\mathbf{C})$, where m is the weighted mean value of selected candidates, σ is the step size and $\mathbf{C}$ is the covariance matrix. The population is evaluated on the objective function and ranked. According to the ranking results parameters m, σ and $\mathbf{C}$ are updated to give more probability to good samples on the next iteration. A small λ accelerates the convergence, while larger values are helpful for noisy functions [13]. The python 'cma' implementation by Hansen et al. [14], was used in our experiments. To fit the general R framework that was used for experimenting, the python code is called via the R to python interface "reticulate" [25].

3 Robotic Application

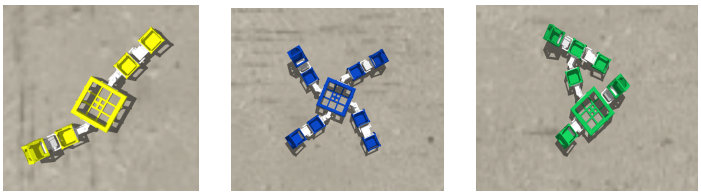
Following the triangle of life framework proposed by [6], we focus on the infancy phase of a robot's life cycle. After a new robot is generated with a specific morphology and controller, it needs to go through a learning stage in order to adapt the configuration of its controller to complete a given task as efficient as possible. Usually this stage is only part of a longer process in which the goal is to find the best morphology-controller combination through evolution. In a real-time real-space scenario this process can be extremely long and costly.

The robot's dynamics, the controller's structure, and, the task a robot needs to complete are variable or unknown factors and can be seen as a black-box function. Given the time every simulation carries it can also be considered expensive. The aim of our optimization task is to find controller configurations that make it possible for the robots to move faster on a restricted number of simulation tests.

The robots are simulated using the **R**obot **Evolve** (Revolve) [16] toolkit. Revolve works on top of Gazebo [3] and incorporates a set of tools to allow an easy definition of the robots, environments to execute the simulations and objective functions to evaluate a robot's performance.

[3] `https://gazebosim.org/`

The robots used in this work are based on the framework RoboGen[4]. The robot's bodies contain three components: A core component housing the robot's microcontroller and battery unit, fixed bricks which allow to attach other components to any of its faces, and lastly, active hinges which are powered by servo motors. Each hinge adds an extra degree of freedom to the robot, thus increasing the input dimensionality in our controller design.



(a) Snake (b) Spider (c) Gecko

Fig. 1: Tested robot morphologies, each simulating an animal structure. The three different body components can be easily distinguished by shape and color. The biggest brick corresponds to the core component, each robot can only have one. Fixed bricks look similar but are smaller in size. Active hinges are illustrated in white.

Three different robot morphologies are tested, each is built to simulate the structure of a snake, spider and gecko respectively. In figure 1 the different robot morphologies and their components are presented. All the servo motors on the robot's body are controlled by an output oscillator. This oscillator depends only on a sinusoid signal determined by three parameters: amplitude, period and phase offset. To reduce the controller's dimensionality the amplitude parameter is fixed to an unit value. Since the tested tasks require the robot to be in constant motion, we assume a fixed amplitude value will not affect the robot's speed as it would in start/stop scenarios.

4 Experiments

The source code of all algorithm configurations, software, and all experiments results presented in this work are freely available for reproducibility at: `https://github.com/frehbach/rebo19a/`.

Two test scenarios are considered to test the viability and performance of the different optimization algorithms. Firstly, the algorithms are extensively benchmarked on the BBOB test suite to assess their performance on varying landscapes and dimensionalities. We simulate the environment of an expensive function by limiting the algorithm's budget to match the amount of permitted

[4] `http://robogen.org`

iterations on the robot application. Secondly, the algorithms are applied to the robot controller problem.

Since the problem can be evaluated in parallel, we do not count the individual objective function evaluations, but rather the sets of parallel evaluations (iterations) done by each algorithm. The machine used for the experiments allows to run six robot simulations efficiently in parallel. Thus, each algorithm can evaluate up to six candidate solutions per iteration.

The different algorithms are started with an initial Latin hypercube design of ten points. q-EI and MOI implementations require the amount of initial samples to be larger than the input dimensionality. Therefore, their amount of initial samples was set to the next multiplier of six which is greater than the respective problem dimensionality. To match the number of available processors the population size for CMA-ES is set to 6 and the initial step size is left at the default value 0.5. After the initial design, each algorithm is given a total of 15 parallel iterations, resulting in a maximum budget of 100 function evaluations. Each experiment is repeated 30 times for statistical analysis.

4.1 First Scenario: BBOB Functions

The Black-Box-Optimization-Benchmarking (BBOB) test suite [12] is one of the most well-known benchmark suites in the evolutionary computation community. The 24 BBOB functions are selected to test the performance of algorithms on landscapes with known difficulty. Every function is scalable to varying input dimensionalities. A detailed description of each function, its optima and properties is available in [17].

In the BBOB suite, each function is available in multiple instances. An instance is a rotated or shifted version of the original objective function. In our experiments the algorithms are run on all 15 standard instances in their five, ten and 20 dimensional version. This will account for possible performance variations in the algorithms caused by problem dimensionality. This is an important point given that the robots can change their morphology and thus reduce or increase the controller dimensionality. All described experiments were run with a recent GitHub version of BBOB, v2.3.1 [15].

4.2 Second Scenerio: Robot controller

Performance on the application case is measured as the maximum speed (m/s) a robot can achieve in a fixed number of function evaluations (simulations). All robots are simulated in a flat world without obstacles. The robot is able to move along the x- and y-axis (ground). A fixed simulation time of 60 seconds is set for each robot.

Two tasks with different degree of difficulty are used, gait learning and directed locomotion. In gait learning the Euclidian distance between the robots starting and end point is measured and divided by the simulation time. The resulting speed is considered to be the robot's fitness.

For directed locomotion, the fitness function takes into account the direction in which the robot moves. Only distance traveled on the y-axis will be measured and then divided by the simulation time.

The input dimensionality of the robots application varies across robots as a results of the different number of active hinge elements. The snake, gecko and spider have 8, 12 and 16 parameters respectively.

Test Problem Publication The software required to run the robot controller application does not run on all platforms. To overcome this issue and to create an open and easy process for everyone to access the robot application, a dockerized version was developed as part of this work. Docker is an open source tool [3] and works on any major operating system. The docker container for running the robot application with a brief usage manual is also available at `https://github.com/frehbach/rebo19a/`.

5 Results and Discussion

All convergence and significance plots for all functions can be found in `https://github.com/frehbach/rebo19a/`.

We do not assume that the collected results are normally distributed. Therefore, we apply non-parametric tests that make less assumptions about the underlying data, as suggested by Derrac et al. [5]. We accept results to be statistically significant if the corresponding p-values are smaller than $\alpha = 0.05$. The Kruskal-Wallis rank-sum test (base-R package 'stats') is used to determine, whether a significant difference is present in the set of applied algorithms or not. If this test is positive, a posthoc test according to Conover (PMCMR R package [18]) for pairwise multiple comparisons is used to check for differences in each algorithm pair. The pairwise comparisons are further used to rank the algorithms on each problem class as follows: The set of algorithms that is never outperformed with statistical significance is considered rank one and removed from the list. This process is repeated until all algorithms are ranked from 1 to 6.

To answer our first research question the initial focus lies on the BBOB test function experiments. First we compare single-core Bayesian optimization (BO) and CMA-ES. The combination of different kernel configurations and acquisition functions explained in Section 2 make up the six tested variants of single-core BO.

BO outperforms CMA-ES on most problems and dimensionalities in the single-core scenario. Figure 2 illustrates this behavior on the ten-dimensional version of the Büche-Rastrigin function, where all BO approaches ranked better than CMA-ES and random search. The different kernel configurations and acquisition functions were observed to have statistically significant difference between each other. Interestingly, greedy acquisition functions, like PV, show better performance than the more popular EI in 16 out of the 24 tested functions. This agrees with results presented in [20]. In the multi-core tests, parallel-BO seems to converge faster and have better performance than CMA-ES on most test

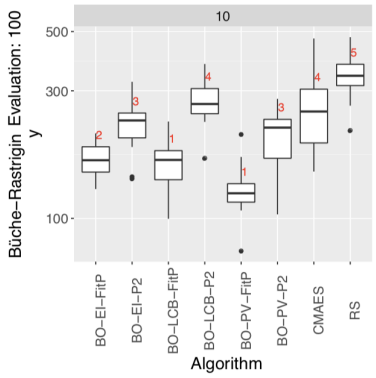


Fig. 2: Single-core BO vs. CMA-ES on the Büche-Rastrigin function. The notation refers to the different combination of acquisition function and kernel configuration. An experiment denoted as BO-PV-P2 then refers to BO using predicted value kernel with parameter $p = 2$. The red number refers to the algorithm rank as determined by the pairwise multiple comparisons test.

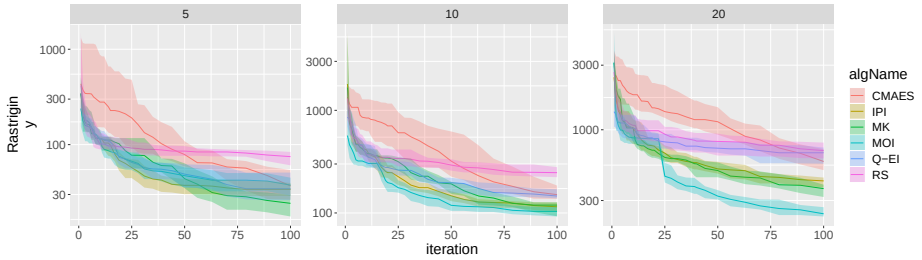
problems. This specially holds true in higher dimensions. Convergence plots for functions Rastrigin, Sharp Ridge and Schwefel in Figure 3 illustrate the performance difference on all tested dimensions. A clear advantage was not always the case for functions with high conditioning or weak global structure. However, only in functions $Weierstrass$ and $Katsura$ did BO not achieved equal or better performance than CMA-ES or random search. Both functions are highly rugged and repetitive.

Our second research question can also be answered by looking at the BBOB results. As shown on Figure 3, q-EI and IPI frequently perform worse than MOI or mK-BO. Based on the single-core BO results it is not surprising that the multi-point adaptation of EI did not perform well. The focus on exploration is disadvantageous on problems with limited function evaluations. This can also be seen for IPI were exploration may win over exploitation.

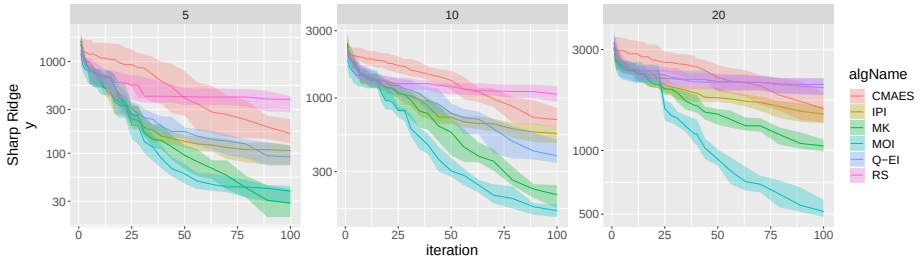
MOI authors remark the algorithm favors exploitative behaviors. If the suggested points tend to exploitation, the performance of MOI is in accordance to the results from single-core experiments, where greedy approaches achieved better values.

For our last research question all algorithms are tested on the expensive black-box robot application. In contrast to the last experiments our aim here is to maximize the objective function. This represents the need to have robots that can move faster.

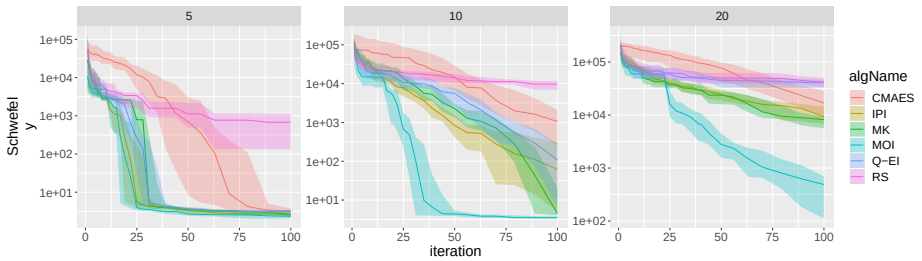
Interestingly there are no clear visible performance differences on some of the robot problems. Random search often performs similarly to all parallel methods. Figure 4 Illustrates the multi-comparison test results for the last iteration in both robot experiments. BO methods rank better than CMA-ES but not always better than random search. If we refer to the benchmark results, a similar be-



(a) Example of a separable multi-modal function



(b) Example of a unimodal function with high conditioning

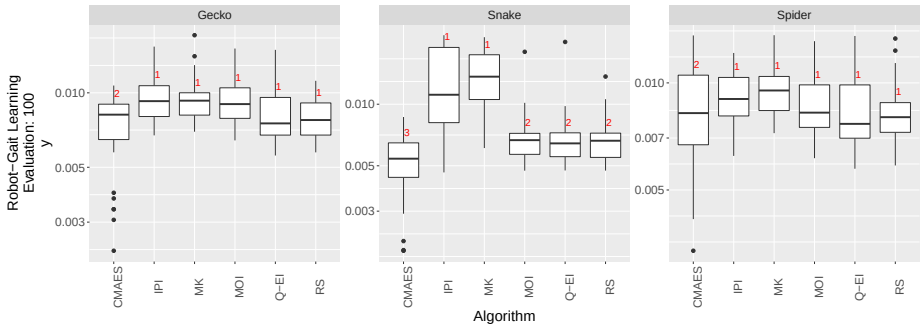


(c) Example of a multi-modal function with weak global structure

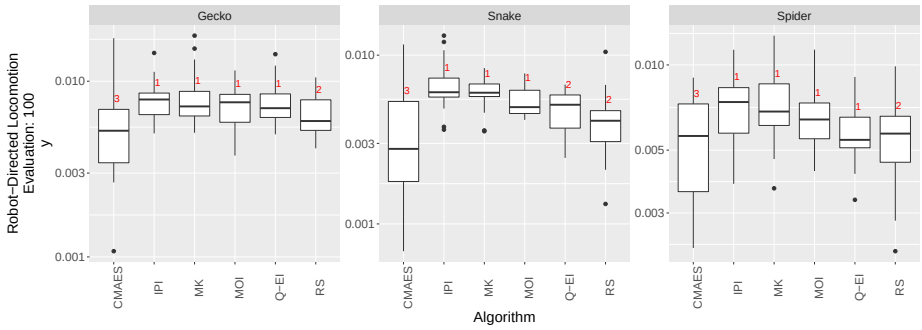
Fig. 3: Parallel algorithms performance on three different BBOB subgroups with different dimensionalities. The convergence plots show the median, upper and lower quartiles. MOI parallel implementation shows a clear advantage in performance for most cases, especially on higher dimensions. Parallel-BO using IPI and q-EI are in many cases outperformed by the simple mK-BO approach.

havior was found for highly multi-modal functions with weak global structures. This is a good indication of the complex controller's landscape and the complex interactions between its parameters.

During the simulations it was possible to visualize the locomotive behaviors of the robots. For the snake it was noticed that it tended to roll in order to achieve greater speeds. This brought an advantage in gait learning but not in directed locomotion. This specific case can be the reason for the performance divergence present for the snake. However, why IPI and mK-BO in particular seem to perform better on this scenario is still open to investigation.



(a) Gait learning



(b) Directed locomotion

Fig. 4: Last iteration ranked multi-comparision test results for the robot application. The red number indicates the algorithm's achieved rank. BO methods maintain a better performance. However similar results by random search suggest BO is not able to efficiently exploit the optimization landscape.

6 Conclusions

To answer our research questions, we compared the performance of sample efficient parallel-BO with the state-of-the-art EA, CMA-ES and random search on expensive black-box problems with a maximum of 100 function evaluations. A first experimental stage was conducted using the BBOB test-functions with the assumption they are expensive to evaluate. As a second stage the algorithms were tested in a real world robot application. To enable easy replicability, a docker container was created including the configuration of all the tested algorithms, experiments and the simulator environment for the robot simulations.

RQ-1 Can parallel variants of BO outperform inherently parallel algorithms like CMA-ES if the evaluation budget is severely constrained? We demonstrated on a varied range of function landscapes that parallel-BO can outperform inherently parallel CMA-ES in problems with very limited number of function evaluations. These results were more clearly seen on higher dimensions. Interestingly we observe that the best performance, is in most cases, achieved by more exploitative acquisition functions (see [20]). As a result of this, Parallel-BO approaches with greedier acquisition function performed better on most experiments.

RQ-2 Which parallel variants of BO show the best performance on which of the tested problem landscapes? Based on the observed behaviour and taking into account that we are dealing with a problem with limited number of function evaluations, our preferred approach is to select parallel-BO with an acquisition function that tends to exploitation. This is the case for MOI and our tested configuration of mK-BO. Of both approaches MOI is our preferred approach for similar applications given the possibility to control its exploration-exploitation trade-off. However, it is necessary to explore possible configuration improvements for both algorithms.

RQ-3 How do BO and CMA-ES compare on the robot application? On the robotic test cases there is no prominent difference in performance. However, the statistical tests support a better or equal performance of the parallel-BO implementations over random search and CMA-ES for all cases. The performance similarity of random search suggests that neither BO nor CMA-ES could take advantage of the objective function landscape. This is an indication that the problem of robot learning implies a highly multi-modal complex objective function.

In summary, the main contributions of this work are: an efficient BO parallelization method, a detailed performance comparison of BO parallelization methods, and a freely available robotics test suite via a docker image for (comparative) challenging problems for optimization methods.

It remains open to further investigate the contribution that each of the different BO configurations had on the final response of mK-BO. Non-stationary kernels for GP can be studied in order to find better configurations for the complex robot task. Furthermore, we are working on developing and testing robot controllers of different structure and re-evaluate the efficiency and efficacy of the optimizers discussed in this paper.

References

1. Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., Lang, M.: mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions (2017)
2. Bischl, B., Wessing, S., Bauer, N., Friedrichs, K., Weihs, C.: Moi-mbo: Multiobjective infill for parallel model-based optimization. In: Learning and Intelligent Optimization (2014)
3. Boettiger, C.: An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review **49**(1), 71–79 (2015)
4. Bongard, J.: Evolutionary Robotics. Communications of the ACM **56**(8), 74–85 (2013)
5. Derrac, J., García, S., Molina, D., Herrera, F.: 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), 3–18 (Mar 2011). https://doi.org/10.1016/j.swevo.2011.02.002
6. Eiben, A., Bredeche, N., Hoogendoorn, M., Stradner, J., Timmis, J., Tyrrell, A., Winfield, A.: The triangle of life: Evolving robots in real-time and real-space (09 2013). https://doi.org/10.7551/978-0-262-31709-2-ch157
7. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary Robotics. In: Siciliano, B. and Khatib, O. (ed.) Handbook of Robotics, pp. 1423–1451. Springer, 1st edn. (2008)
8. Forrester, A., Sobester, A., Keane, A.: Engineering design via surrogate modelling: a practical guide. John Wiley & Sons (2008)
9. Frazier, P.I.: A tutorial on bayesian optimization (2018)
10. Ginsbourger, D., Le Riche, R., Carraro, L.: Kriging Is Well-Suited to Parallelize Optimization, pp. 131–162. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-10701-6_6, `https://doi.org/10.1007/978-3-642-10701-6_6`
11. Haftka, R.T., Villanueva, D., Chaudhuri, A.: Parallel surrogate-assisted global optimization with expensive functions–a survey. Structural and Multidisciplinary Optimization **54**(1), 3–13 (2016)
12. Hansen, N., Auger, A., Mersmann, O., Tusar, T., Brockhoff, D.: COCO: A platform for comparing continuous optimizers in a black-box setting. ArXiv e-prints (Aug 2016)
13. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**(2), 159–195 (2001)
14. Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634 (Feb 2019). https://doi.org/10.5281/zenodo.2559634, `https://doi.org/10.5281/zenodo.2559634`
15. Hansen, N., Brockhoff, D., Mersmann, O., Tusar, T., Tusar, D., ElHara, O.A., Sampaio, P.R., Atamna, A., Varelas, K., Batu, U., Nguyen, D.M., Matzner, F., Auger, A.: COmparing Continuous Optimizers: numbbo/COCO on Github (Mar 2019). https://doi.org/10.5281/zenodo.2594848, `https://doi.org/10.5281/zenodo.2594848`
16. Hupkes, E., Jelisavcic, M., Eiben, A.E.: Revolve: A versatile simulator for online robot evolution. In: Sim, K., Kaufmann, P. (eds.) Applications of Evolutionary Computation. pp. 687–702. Springer International Publishing, Cham (2018)
17. Nikolaus, H., Steffen, F., Raymond, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Research Report inria- 00362633v2, INRIA (2009)

18. Pohlert, T.: The pairwise multiple comparison of mean ranks package (pmcmr) (2014), http://CRAN.R-project.org/package=PMCMR, accessed on Jan. 12, 2016
19. Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning, MIT Press, Cambridge, MA, USA (Jan 2006)
20. Rehbach, F., Zaefferer, M., Naujoks, B., Bartz-Beielstein, T.: Expected improvement versus predicted value in surrogate-based optimization (2020)
21. Roustant, O., Ginsbourger, D., Deville, Y.: DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. Journal of Statistical Software **51**(1), 1–55 (2012)
22. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (Jan 2016). https://doi.org/10.1109/JPROC.2015.2494218
23. Srinivas, N., Krause, A., Kakade, S.M., Seeger, M.W.: Information-theoretic regret bounds for gaussian process optimization in the bandit setting. IEEE Transactions on Information Theory **58**(5), 3250–3265 (May 2012). https://doi.org/10.1109/tit.2011.2182033, http://dx.doi.org/10.1109/TIT.2011.2182033
24. Ursem, R.K.: From expected improvement to investment portfolio improvement: Spreading the risk in kriging-based optimization. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) Parallel Problem Solving from Nature – PPSN XIII. pp. 362–372. Springer International Publishing, Cham (2014)
25. Ushey, K., Allaire, J., Tang, Y.: reticulate: Interface to 'Python' (2019), https://CRAN.R-project.org/package=reticulate, r package version 1.13