MVA Reinforcement Learning
Optimization of very difficult functions

Nathan de Lara, Florian Tilquin

January 14, 2016

1 Introduction

1.1 Problem statement

The goal of this paper is to test and compare recently developed algorithms for the optimization of *very difficult functions*. This work is based on the papers by Bubeck [2], Grill [4], Lazaric [1], Bull [3] and Valko [5]. Each one of this paper has a specific definition of *difficult* but the general idea is that the function to optimize has many local maxima and only one global maximum, it has very fast variations and is not necessarily differentiable such that a gradient-based approach to find the optimum should not be successful. All functions are assumed to be bounded and to have a compact support which, up to scaling can be fixed to be $[0, 1]^p$. In this paper, we only consider $p = 1$. In the end, the general formulation of the problem is:

$$\text{maximize } f(x) \text{ for } x \in [0, 1] \tag{1}$$

1.2 About the multi-armed bandit

As previously mentioned, a gradient-based approach is not likely to perform well on the considered functions. Thus, the idea is to use a multi-armed bandit with theoretically an infinite number of arms, sometimes called *continuum-armed bandit*. The algorithms progressively defines a sequence of evaluation points x_t and observes a reward $y_t = f(x_t) + \xi_t$ where ξ_t is a noise term. The choice of x_{t+1} depends on the sequence of $(y_{t'})_{t' \leq t}$. Depending on the algorithm, it is meant to converge to x^* while controlling the cumulative regret:

$$R_T = \sum_{t=1}^{T} f(x^*) - f(x_t) \tag{2}$$

Or just simply ensuring that a good estimate of x^* is present in the sequence (i.e minimizing simple regret). They can also be designed to be robust to noise in function estimations or correlated feedback. This design depends on the optimization context: is it an off-line or online optimization ? how long does an evaluation of the function take? etc.
In practice, for the algorithms considered, $x_t = \mathcal{U}(I_t)$. This means that x_t is pulled uniformly at random in an interval I_t that is chosen by the algorithm. The

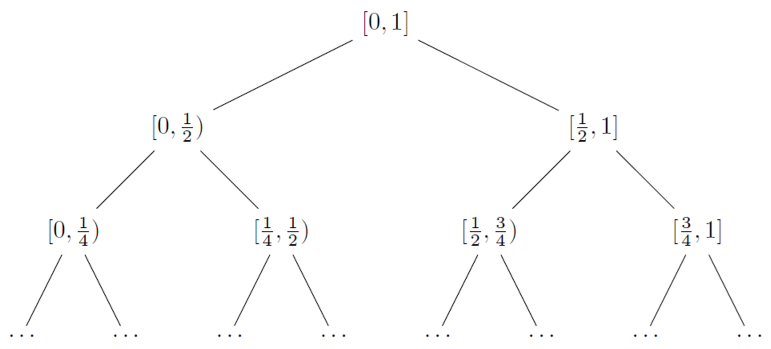


Figure 1: A dyadic tree from [3]

set of intervals defines a tree structure such that the set of leaves is a partition of $[0, 1]$. In the end, the original problem is transformed into a particular case of regular multi-armed bandit. Without any prior information on the function to optimize, we only consider here the family of *dyadic trees*. As mentioned in [3], the dyadic tree on [0, 1) is the tree with root node [0, 1), and where each node [a, b) has children $[a, \frac{1}{2}(a+b), [\frac{1}{2}(a+b), b)$. An example of dyadic tree is displayed in 1.2.

1.3 Notations

In the rest of this paper, we note:

- h, i the coordinates of a node in the dyadic tree such that h is the depth of the node and i its width
- $\widehat{\mu}_{h,i}$ the empirical estimate of the reward associated to the node h, i
- $n_{h,i}$ the number of times the node h, i has been hit during the evaluations up to the current time
- T_{max} the total number of evaluations of the function allowed or *budget*

2 Algorithms

In this section, we list the algorithms to be compared and briefly present their respective behaviors. The reader is welcome to consult the original papers for more detailed descriptions and proofs on theoretical performances. As previously mentioned, each algorithm defines a sequence of intervals I_t from which x_t is sampled. Typically, the selected interval maximizes a certain selection function of $(x_{t'}, y_{t'})_{t'<t}$ among a subset of considered intervals. The main differences in the following algorithms is the definition of the selection function, which is most of the time an upper bound on the expected reward of the arm and the choice of the considered intervals at each time step, which is sometimes called "expansion rule".

2.1 Hierarchical and Parallel Optimistic Optimization

This algorithm is called *Optimistic* because its idea is to sequentially building a tree and try the most promising children.

Upper bounds $B_{h,i} = min(U_{h,i}, max(B_{h+1,2i-1}, B_{h,2i}))$ where:

$$U_{h,i} = \widehat{\mu}_{h,i} + \nu\rho^h + \sqrt{\frac{2\log(T_{max})}{n_{h,i}}} \tag{3}$$

Update rule At each time step, the most promising child with respect to B is added to the tree.

Parallel Optimistic Optimization When there is no prior knowledge on the function to optimize, the tuning of the parameters ν and ρ of equation 3 can be difficult. Thus, the idea of this algorithm is to sequentially launch several HOOs that keep running in parallel in order to get the best parameters. Note that this algorithm can be run with other algorithms than HOO such as HCT.

2.2 High-Confidence Tree

This algorithm exists in two different versions: with correlated and uncorrelated feedback. For the purpose of performance comparison with other algorithms, we only consider the uncorrelated feedback, which is close to HOO algorithm. In order to be pulled, an arm must maximize B among the arms that have not been pulled enough yet. In order to reduce computational cost, U is refreshed only when t is a power of 2.

Upper bounds $B_{h,i} = min(U_{h,i}, max(B_{h+1,2i-1}, B_{h,2i}))$ where:

$$U_{h,i} = \widehat{\mu}_{h,i} + \nu\rho^h + \sqrt{\frac{c^2\log(1/min(1, \frac{c_1\delta}{2^{\lfloor\log(t)\rfloor+1}}))}{n_{h,i}}} \tag{4}$$

Update rule Only the leaves of the current covering tree that have not been pulled enough with respect to a certain threshold $\tau_h(t)$ are expanded.

2.3 Stochastic Simultaneous Optimistic Optimization

This algorithm is called *simultaneous* because it can perform at each time steps as many evaluations as the depth of its current covering tree. Indeed, the idea is that, in order not to rush into a local maximum, it is good to keep sampling from small depths until the estimation of the local reward is better known. Each node a its own evaluation budget k in order no to spend to much budget on the first nodes.

Upper bounds In order to be pulled, a leaf must both have a positive remaining individual budget and maximize B among the leaves of the same depth:

$$B_{h,i} = \widehat{\mu}_{h,i} + \sqrt{\frac{\log(\frac{T_{max}k}{\delta})}{2n_{h,i}}} \tag{5}$$

Update rule Once a node that maximizes B has used its entire budget, it is expanded.

2.4 Adaptive-Treed Bandits

This algorithm has a slightly different spirit than the others. The tree is not sequentially built but given as an input and the evaluations are performed among a set of *active boxes* or *active nodes* (which in dimension 1 are simply intervals). Knowing the entire tree allows to update the statistics of the children of active nodes each time an arm is pulled, while in the other algorithms, only the statistics of the parents could be updated.

Upper bounds At each time step, the arm pulled is chosen among the active nodes and maximizes:

$$B_{h,i} = \widehat{\mu}_{h,i} + (1+2\nu)r_{h,i} \tag{6}$$

where $r_{h,i} = 2\sqrt{\dfrac{\log[2^{h+1}(\tau + n_{h,i})]}{n_{h,i}}}$ is the confidence radius of the interval.

Update rule If an active node has a radius small enough compared to the ones of its children, it is removed and replaced by them.

3 Results

3.1 Experimental Setup

Objective functions We test the algorithms on different reference functions from [5] and [4]:

1. Two-sine product function: $f_1(x) = \frac{1}{2}(\sin(13x).\sin(27x)) + 0.5$.
2. Garland function: $f_2(x) = 4x(1-x).(\frac{3}{4} + \frac{1}{4}(1 - \sqrt{|\sin(60x)|}))$.
3. Grill function: $f_3(x) = s(\log_2(|x{-}0.5|).(\sqrt{|x-0.5|}-(x{-}0.5)^2)-\sqrt{|x-0.5|}$ where $s(x) = \mathbf{1}(x - \lfloor x \rfloor \in [0, 0.5])$.

The associated plots are displayed in 3.1. It is important to have in mind that sampling a point at random for this functions would provide an average regret of approximately 0.46, 0.46 and 0.32 respectively.

Algorithms setup In order to compare the performances of the different algorithms, we set a total number of function evaluations T_{max} and a number of runs N. Then we compute for each and each algorithm run the best value returned $\widehat{x}^*$. Besides, we keep track of all the points sampled during each run and their associated reward in order to check the concentration of evaluations around the maximums and to compute quantities of interest such as the cumulative regret defined in 2, the average cumulative regret or the best estimate up to time t. The tuning of the parameters specific to each algorithm is performed manually. The algorithms are tested without noise and with different Gaussian noises.

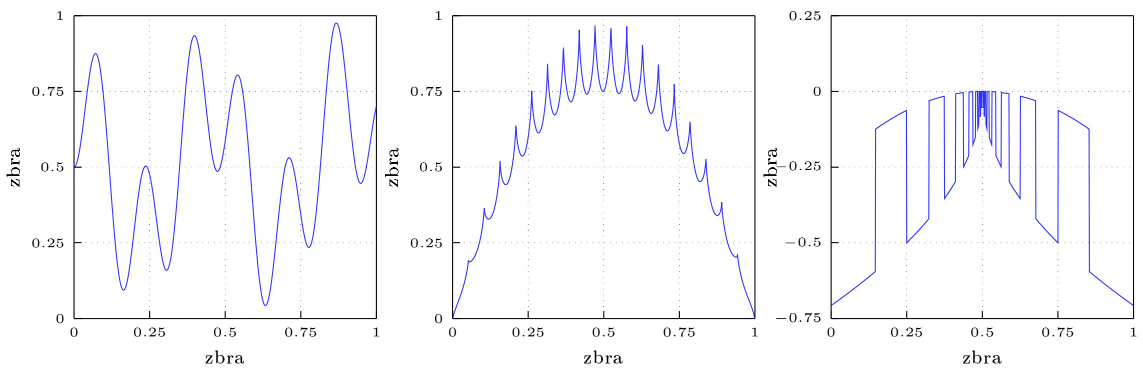


Figure 2: From the left to the right: Two-sine product, Garlang, Grill.

As mentioned in section 1.2, the algorithms have different objectives (minimizing cumulative or simple regret for example), thus one must evaluate the performances according to each design.

3.2 Analysis

On figure 3.2 we can see the different sampling strategies of the algorithms. For HOO, we see a progressive dichotomy of the space that ends up in a local maximum. This explains the distinct columns in POO that correspond to different local maximums, fortunately, only the best is kept in the end. For HCT, we see that the sampling is progressively concentrated in the three most promising areas with a higher concentration for the one on the right that actually contains x^*. The "jumps" in the sampling correspond to the refresh of all bounds that happens when t is a power of 2, this triggers a new exploration phase in order to avoid being trapped into local minimums like it can happen for HOO. Finally we see that StoSOO maintains a higher exploration rate, indeed, this algorithm focuses on simple and not cumulative regret. But still, it is clear that more sampling is performed in the areas of local maximums.

Figure 3.2 presents the cumulative regrets of the algorithms. The more difficult the function, the higher the regret. Besides, depending on the level of exploration of the algorithm, we see different increase speed. For example, HOO stabilizes faster than the other algorithms thus after a certain numbers of steps, its cumulative regret is either constant or linear. On the contrary, SOO always maintains exploration and has consequently a higher cumulative regret.

Figure 3.2 presents the evolution of the best estimate up to time t under noisy evaluations. Here HOO does not perform as well as the other algorithms because it explores less and is then more likely to "trust" bad function estimations.

Finally, figure 3.2 present the results for ATB algorithm. As their is no recorded testing of this algorithm, we cannot confirm that our results matches its expected behavior. Besides, this algorithm turned out to be extremely sensitive to parameters settings, going from random to brute force exploration. However, with tuned parameters it achieves better cumulative regret than a random or brute force algorithm and samples more around local maximums, even-though it does not always find the global one.

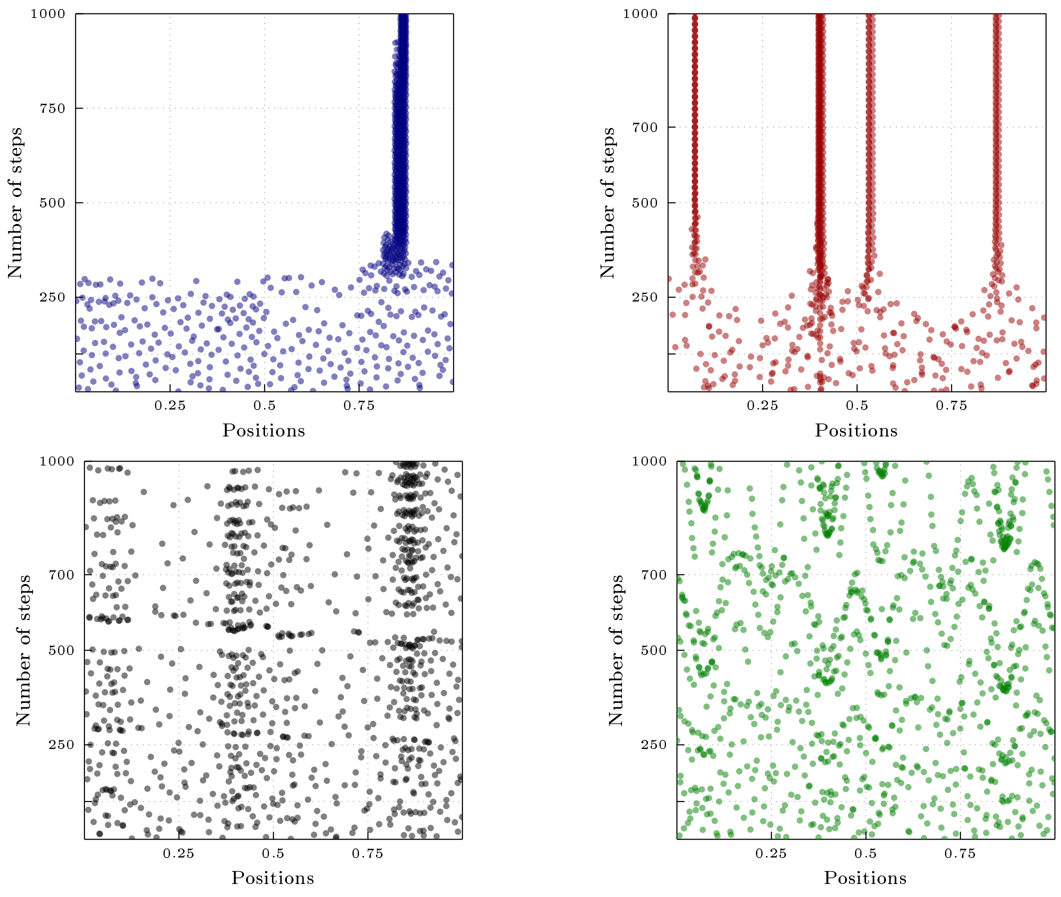


Figure 3: Points sampled by the algorithms for Sinprod with 1000 evaluations. From top left to bottom right : HOO, POO, StoSOO and HCT. For this function, $x^* \simeq 0.86$ and the second best point is roughly in 0.39.

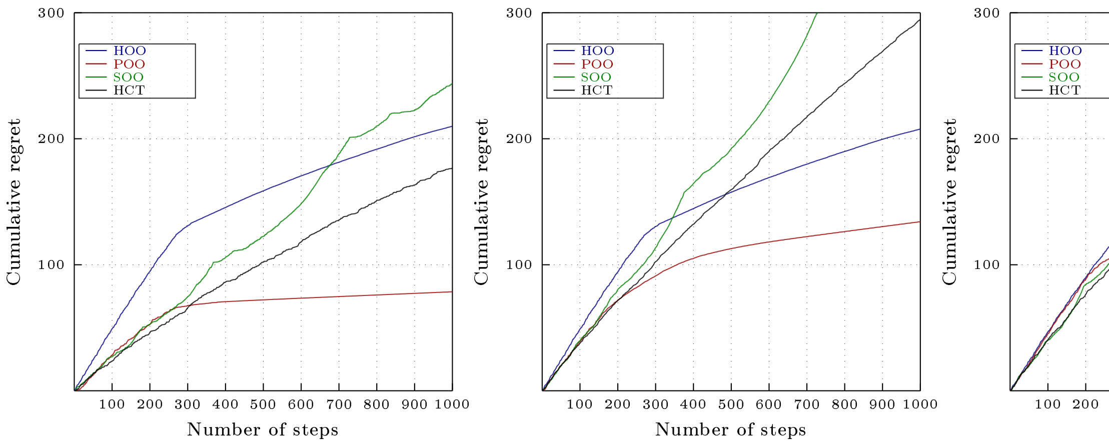
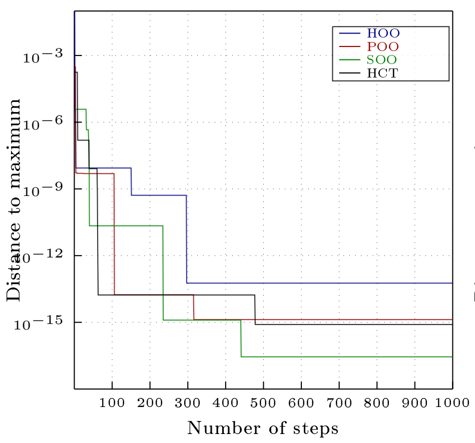
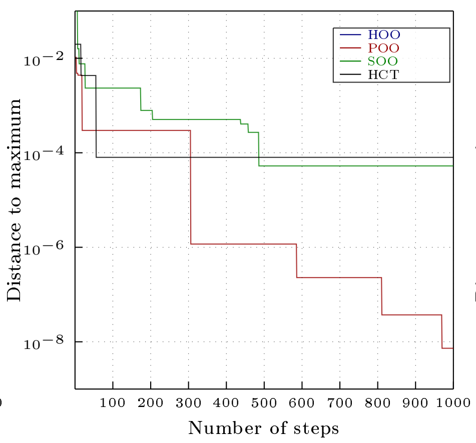
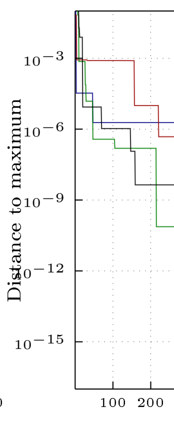
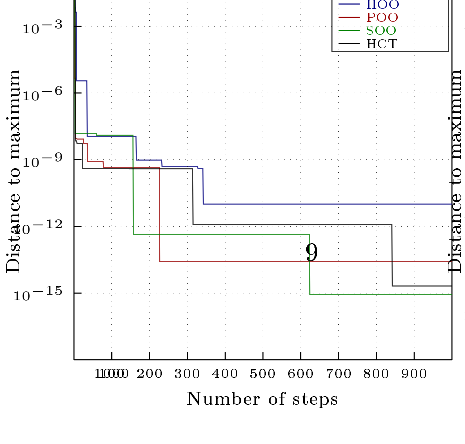


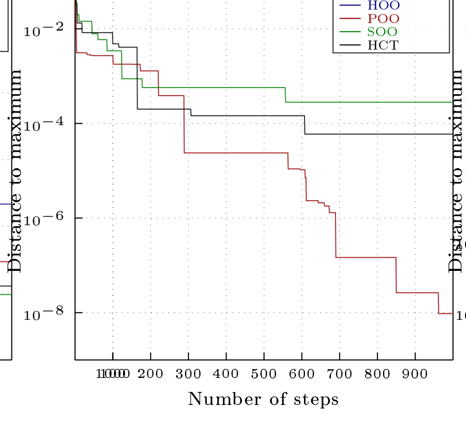
Figure 4: The cumulative regret of the 5 algorithms on the three difficult functions, with 1000 evaluations. From left to right : Grill function, Garland function, Sinprod function.

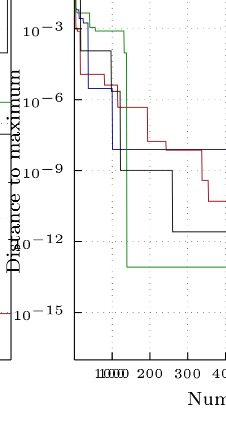

HOO
POO
SOO
HCT
Distance to maximum
Number of steps


HOO
POO
SOO
HCT
Distance to maximum
Number of steps


Distance to maximum


HOO
POO
SOO
HCT
Distance to maximum
Number of steps


HOO
POO
SOO
HCT
Distance to maximum
Number of steps


Distance to maximum

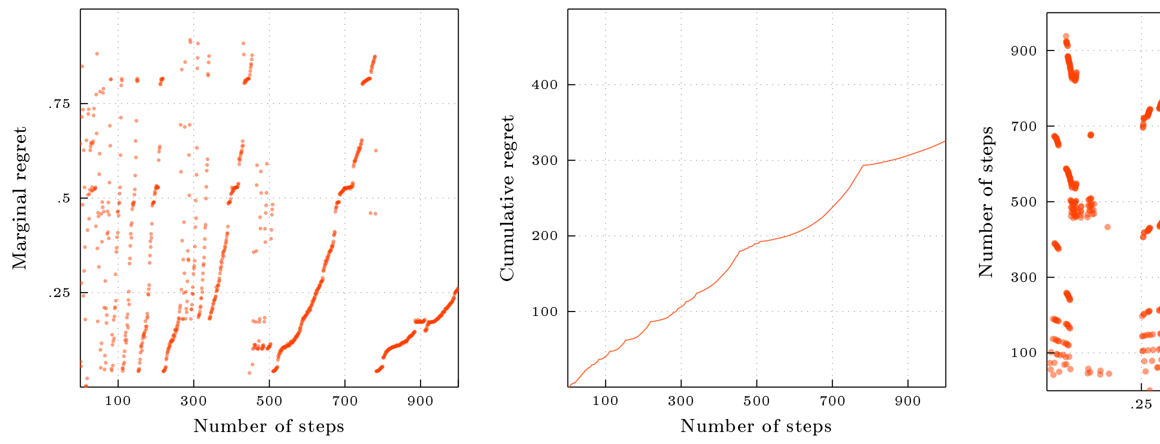


Figure 6: From the left to the right: simple and cumulative regrets and sampled points for ATB algorithm on Sinprod optimization.

References

[1] Mohammad Gheshlaghi Azar, Alessandro Lazaric, and Emma Brunskill. Online stochastic optimization under correlated bandit feedback. *arXiv preprint arXiv:1402.0562*, 2014.

[2] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvari. X-armed bandits. *The Journal of Machine Learning Research*, 12:1655–1695, 2011.

[3] Adam D Bull. Adaptive-treed bandits. *arXiv preprint arXiv:1302.2489*, 2013.

[4] Jean-Bastien Grill, Michal Valko, and Rémi Munos. Black-box optimization of noisy functions with unknown smoothness. In *Neural Information Processing Systems*, 2015.

[5] Michal Valko, Alexandra Carpentier, and Rémi Munos. Stochastic simultaneous optimistic optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 19–27, 2013.