Bandits attack function optimization

Philippe Preux and Rémi Munos and Michal Valko

Abstract— **We consider function optimization as a sequential decision making problem under the budget constraint. Such constraint limits the number of objective function evaluations allowed during the optimization. We consider an algorithm inspired by a continuous version of a multi-armed bandit problem which attacks this optimization problem by solving the tradeoff between exploration (initial quasi-uniform search of the domain) and exploitation (local optimization around the potentially global maxima). We introduce the so-called Simultaneous Optimistic Optimization (SOO), a deterministic algorithm that works by domain partitioning. The benefit of such an approach are the guarantees on the returned solution and the numerical efficiency of the algorithm. We present this machine learning rooted approach to optimization, and provide the empirical assessment of SOO on the CEC'2014 competition on single objective real-parameter numerical optimization test-suite.**

I. Introduction

FUNCTION optimization has been a recurring topic for centuries. In this work, we assume that the function is available only through a black-box: one provides a point to the black-box which returns the value of the function at that point. It is customary to make assumptions about the objective function (the function being optimized), whether it is continuity, smoothness, or even differentiability. These assumptions are not necessarily met by the real-world objective functions and commonly they are impossible to verify. Moreover, the objective function may be nondeterministic, returning different values along time for the same point. An important practical aspect is that the evaluation of the objective function at a given point always takes some resource (computational time, energy, bandwidth, money, ...) and it is necessary to optimize a trade-off between the quality of the optimum being found and the amount of resources that have been used to find it.

Most of real-world functions to optimize are multi-modal. One is then interested in finding a *global optimum*, or one among them if there are many optimal points. We distinguish *global* from *local* numerical optimization algorithms. A "global" optimization algorithm is guaranteed to return a global optimum given enough resources, whereas a "local" optimization algorithm is only guaranteed to return a local optimum, whatever the computational resource being used. Therefore, we require an asymptotic guarantee of optimality. More interestingly, we aim for a finite-time guarantee, where we provably evaluate the quality of the returned solution as a function of the evaluation budget.

Philippe Preux is with Université de Lille, LIFL (UMR 8022 CNRS), and INRIA, France. Rémi Munos and Michal Valko are with INRIA, and LIFL, Lille, France. (email: {philippe.preux,remi.munos,michal.valko}@inria.fr). This work was supported by the French Ministry of Higher Education and Research, Nord-Pas-de-Calais Regional Council, and FEDER through the "Contrat de Projets Etat Region (CPER) 2007-2013", and by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 231495 (project CompLACS).

With no prior knowledge on the function such a global optimization algorithm would perform iteratively the following action: based on already evaluated points and their values, choose the next point to evaluate. Deciding on an action based on past actions and their consequences is known as *sequential decision making*. Since the objective function is unknown, this problem is *under uncertainty*. There is a vast body of literature on this problem, coming from the various scientific domains such as machine learning, statistics, and operation research. In this paper, we wish to build on the existing body of work and bring a principled approach to function optimization. Hence, we introduce the necessary background to derive an algorithm which is fairly simple but performs well in practice and which properties can be studied theoretically. We regard the main contribution of this paper as bringing the idea that function optimization is a sequential decision making problem, a view that opens new ways of approaching function optimization.

In the rest of this paper, we first introduce the framework of sequential decision making under uncertainty and state of the art concepts that will be used to design a function optimization algorithm. Next, we introduce our algorithm acronym-ed as SOO and discuss some of its properties. In section IV, we perform the experimental assessment of SOO based on the *CEC'2014 competition on real parameter numerical optimization* test suite. We believe that the results show that SOO is a really serious alternative for function optimization. Based on these results, we discuss the strengths and weaknesses of SOO and its application.

II. Function optimization and sequential decision making under uncertainty

As the background of this work stems from the "sequential decision making problem under uncertainty", we first describe the necessary basics.

Imagine an agent interacting with its environment; we assume that time is discrete. The agent perceives its environment in some way and finds itself in a *state*; the agent possesses a certain repertoire of actions; at each time step – it is a *sequential* process – the agent decides which action to perform based on its previous history, in order to reach a certain goal. The goal is formalized by an objective function which is not known to the agent: the agent somehow probes the objective function through its actions but not directly through the objective function. When the agent performs an action, it receives a consequence, also known as a return or a *reward*; this consequence reflects the value of the action with regards to the objective function.

Let us formalize the setting above: an agent faces a set of K actions. Each action a_k is associated to a certain distribution ν_k, whose expectation is denoted by $\mu_k = E[\nu_k]$. At any round t, the agent executes an action $a_t \in \{1, \dots K\}$ and receives a noisy sample r_t –called reward– drawn (independently) from the distribution ν_{k_t} (thus we have $\mathbb{E}[r_t] = \mu_{k_t}$). The goal of the agent is to find the strategy that maximizes the sum of collected rewards in expectation. This problem is known as the "multi-armed bandit problem", introduced in 1933 by W.R. Thompson [14] and independently in 1952 by H. Robbins [12]. This is also known as the exploration vs. exploitation dilemma. A related question is: how can the agent identify as fast as possible the best action, that is the one which is the most rewarding on average ($k^* = \arg\max_k \mu_k$). This has been called the simple regret problem or best arm identification in multi-armed bandit settings [3], [1].

This problem has witnessed remarkable advances within the last 15 years thanks to the introduction of the notion of *upper confidence bounds* [2]. We illustrate the approach with a simple example: suppose the agent faces $K = 2$ possible actions; at time $t = 15$, the action 1 has been executed by the agent 5 times, with an average reward of 0.3, while 2 has been executed 10 times resulting in the same average rewards of 0.4: at the next time step, which action should the agent perform? There are many strategies to cope with this situation, well-known in the EC community (ε-greedy, softmax, Boltzmann-Gibbs, ...). Here we consider the "Upper-Confidence Bound" (UCB) approach that follows the so-called "optimism in the face of uncertainty" principle, which consists in selecting the action that has the highest rewards in all possible environments that are reasonably compatible with the observations (i.e. rewards observed so far). Here, action 2 seems better than 1 in terms of average empirical rewards. However action 1 has been chosen less often than 2, thus the uncertainty over its true mean μ_1 is higher than for μ_2, thus the set of possible values of μ_1 given the observed rewards is potentially larger than for action 2. Here, exploitation would mean selecting action 2 since its empirical mean is higher than 1, but exploration would imply selecting 1 in order to get additional information about action 1 and thus reduce the uncertainty over μ_1. The UCB algorithm solves this exploration-exploitation tradeoff by defining an UCB on the mean of each action based on the rewards observed so far and the number of times each action have been chosen. Then the action with highest UCB is selected.

To be more specific, assume that each action k has been executed $n_{k,t}$ times up to time t and the average reward observed on this action is $\hat{\mu}_{k,t}$. Then, the UCB strategy selects the action k_t that maximizes the following UCBs:

$$k_t = \arg \max_{1 \leq k \leq K} \left(\hat{\mu}_{k,t} + \sqrt{\frac{2 \log t}{n_{k,t}}} \right) \quad (1)$$

One can show that the choice of UCB leads to an expected cumulative performance (sum of obtained rewards) up to time n which is almost as high as the optimal value (which would be $n \max_k \mu_k$, *i.e.* n times the mean of the best action) if we knew the action distributions. Also, following the UCB algorithm, one may prove that the number of time steps at which suboptimal actions are executed grows only logarithmically with time, that is the exploration cost grows logarithmically with time, which is optimal in this setting (there exist lower-bounds saying that one cannot expect to make less than a logarithmic number of mistakes, see [7]).

In the related (simple regret or best-arm identification) setting, one is given a fixed budget n of action executions, and the objective is to explore as efficiently as possible the set of actions during n rounds and make a final recommendation of what is the best action. This formulation is closer in spirit to the problem of function optimization under budget constraint and shares strong links (in terms of exploration-exploitation tradeoff) with the previous problem of maximizing the sum of rewards, see [3], [1].

III. Simultaneous Optimistic Optimization

A. The algorithm

The application of the previous framework to function approximation over a continuous domain is interesting. The set of possible actions is the domain of definition of the function to optimize (so that one action is one point of the domain) and the reward subsequent to an action execution is the value of the objective function at the point associated to the action.

At first, the reader may raise serious concern about the fact that the set of possible actions is hence uncountable. It turns out that this problem can be overcome, both in theory and in practice, by relying on some smoothness assumption of the objective function. This setting falls in the category of "structured bandit problems". Of course, one would like to set minimal assumptions on the target function. However the optimization problem is known to suffer from the curse of dimension, even in finite spaces. Also the no-free-lunch theorem somehow tells us that for any clever algorithm, there exists a problem on which the clever algorithm performs poorer than a stupid algorithm (say an algorithm that just explores uniformly at random). However if we restrict the classes of functions of interest, then it is possible to define algorithms that would be good on such classes. Now, ideally we would like to assume some smoothness on the objective function, but without having to tune the algorithm according to the specific smoothness of the function (which is usually unknown).

This is the reason we rely on an algorithm called "Simultaneous Optimistic Optimization" (SOO), which has been introduced by Munos in 2011 [9]. SOO is a deterministic algorithm, meant to optimize deterministic functions which are assumed to be locally smooth near their global optima (in a specific sense) but where the actual smoothness does not need to be known. The actual smoothness is not needed by the algorithm, but the performance of the algorithm will depend on this smoothness.

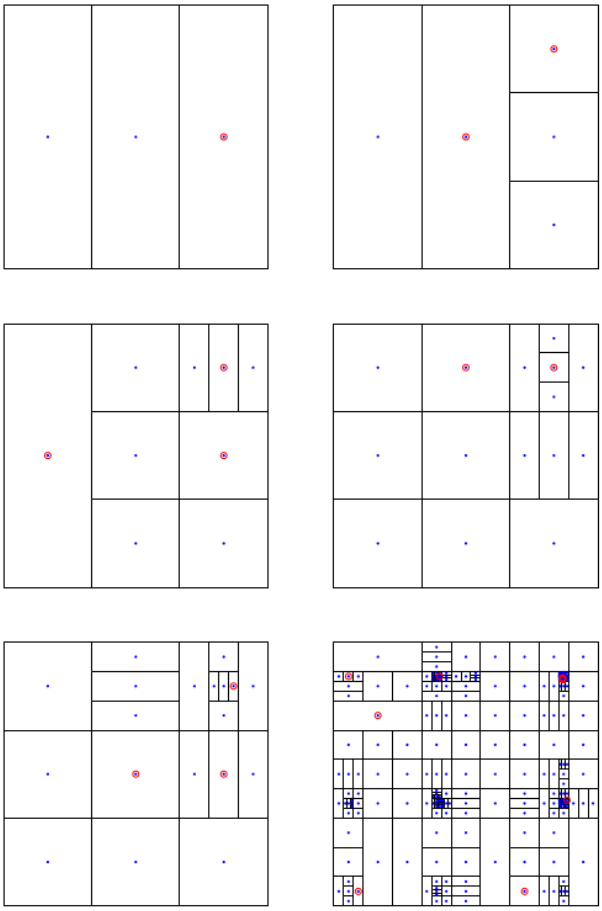

Fig. 1. An illustration of SOO at work during a function optimization. Assuming a square bi-dimensional domain of search for the optimum and $S = 3$, the first steps splits the initial cell into S sub-cells (top left); the split occurs along one dimension. The next step splits one of those sub-cells (top right), and so on and so forth. Steps 1 to 5 are represented from top to bottom. Bottom right is the partition obtained after 150 such steps on a certain function. At each step, the cells being split have their center circled in red. SOO progressively concentrates on the most promising cells.

SOO works by partitioning the domain of definition of the objective function. At first, the whole domain of search is one cell. This cell (as well as all forthcoming cells) is represented by the point at its center, and the value of the objective function for this point. The first iteration consists in splitting this cell into S sub-cells; the cut occurs along one dimension. The center point of each sub-cell is evaluated. Assuming we minimize the objective function, SOO chooses to split the cell with the smallest value at its center; the split then occurs along another dimension (with regards to the initial split), and the center point of each of these sub-cells is evaluated. By so doing, SOO builds a tree of cells, the root being the entire search domain. Then, at each iteration, SOO considers each depth in the tree and for each depth, selects the cell with lowest value at this depth and splits this cell *only* if its value is lower than all previously selected cells of lower depths. The iterations go on until the budget of function evaluations is exhausted. The point with lowest evaluation is then returned. Fig. 1 illustrates the procedure. Taking S odd, the cell being split shares its center with one of the sub-cells so that each cell split involves $S-1$ objective function evaluations. That way, SOO performs adaptive partitioning, focusing on the area of the domain which is the most promising at each iteration.

SOO is sketched in Algorithm 1.

B. Properties of SOO

- SOO is very simple, hence very easy to implement in a very efficient manner.
- SOO is a rank-based algorithm (the actual values of f do not matter, but only their pairwise comparisons).
- SOO does not rely on the knowledge of the smoothness of the objective function.
- SOO is consistent: it converges asymptotically towards the global optimum.
- after t time steps, the expected difference between the value of the global optimum of the objective function $f^* = \min_{x \in X} f(x)$ and the value of the best point x_t returned by SOO:

$$f(x_t) - f^* \text{ is decreasing in } O(t^{-1/d}), \tag{2}$$

where d is the near-optimality dimension of f (defined precisely in [9]), which provides a measure of the quantity of near-optimal points near the global optimum of f. Note that exponential rate $O(e^{-ct})$ can also be achieved in the non-trivial case $d = 0$. This is a nice result and SOO is unique for being a global optimization algorithm for which such a finite-time performance result have been formally proven under such weak assumption on the objective function.

C. Related algorithms

SOO is a global function optimization algorithm; it may be related to many other algorithms, in particular with ES, CMA-ES, and other evolutionary algorithms. However, SOO is deterministic. Actually, SOO is very closely related to the algorithm DiRect introduced by Jones *et al.* in 1991 [6]. DiRect stands for "Dividing Rectangles". DiRect assumes the objective function to be globally Lipschitz (with unknown Lipschitz constant) whereas SOO makes a much weaker assumption on f which is only an assumption around the global optimum. In addition, finite-time performance guarantees are obtained for SOO whereas DiRect only enjoys an (asymptotic) consistency result.

IV. Experimental evaluation of SOO

The design of SOO stems from research in bandit theory which aims at proposing algorithms which are amenable to a theoretical analysis of their performance. Actually, to the best of our knowledge, SOO is the only existing global function optimization algorithm for which the discrepancy between the best found point after N function evaluations and the optimum is known under very weak assumptions. That being said on the theoretical basis, we were eager to assess its performance in practice.

Algorithm 1 Sketch of algorithm SOO to minimize function f in a given domain using maxeval calls to the objective function. Parameters are the maximum depth of the tree h_{max}, the number of function evalautions maxeval, and the split factor k.

$t \leftarrow 1$ // number of objective function evaluation having been made so far
// $\mathcal{C}$ denotes the set of cells:
// initially, there is one cell that comprises the whole domain of definition of the objective function, at depth 0 in the tree
$\mathcal{C} \leftarrow \{(\text{whole domain}, 0)\}$
while $t \leq$ maxeval **do**
  $\mathcal{B} \leftarrow \emptyset$ // set of cells that have to be splitted
  $v_{min} \leftarrow +\infty$ // current best evaluation of the function
  for $h = 0; h < min(maxdepth(\mathcal{C}), h_{max})$ **do**
    among all leaves of $\mathcal{C}$ at depth h, select the one associated to the best point: x_h^*
    if $f(x_h^*) \leq v_{min}$ **then**
      add this cell to $\mathcal{B}$
      $v_{min} \leftarrow f(x_h^*)$
    end if
  end for
  split each cell in $\mathcal{B}$ into k sub-cells and add these sub-cells to $\mathcal{C}$; evaluate their center point; update t accordingly (exit the loop if maxeval is reached)
end while
Return (the best found point)

A. Set-up

The setup of the experimental evaluation of SOO is that of the CEC'2014 competition on single objective real-parameter numerical optimization (see [8]). 30 objective functions should be optimized in 10, 30, 50, and 100 dimensions with a given budget of function evaluations. The budget is proportional to the dimension: $10^4\times$ the dimension, hence 10^5, 3×10^5, 5×10^5, and 10^6 function evaluations. Unless otherwise stated, all results are provided within this setup. Objective functions are numbered from 1 to 30, and the optimum value is 100 times the number of the function (that is, 100 for function 1, up to 3000 for function 30). Functions 1 to 16 may be said to be atomic, whereas functions 17 to 22 are sums of 3, 4 or 5 atomic functions, and functions 23 to 30 are weighted sums of 3, 4 or 5 atomic functions.

B. SOO parameters

SOO has just a few parameters:

- S: the number of cells resulting from the split of a cell.
- the maximum depth of the tree h_{max} (as a function of N)
- the direction of split of cells.

We did a little exploration of parameter values and ended up using default values since the changes in performance were not significant.

The selected cell is thus split into $S = 3$ sub-cells. The maximum depth of the tree is $h_{max} = 10 \times \sqrt{(\log N)^3}$ (set taking inspiration from *cf.* [15] and testing various values experimentally). The dimensions being numbered, the direction of a split is merely the next dimension with regards to the previous one.

TABLE I
MEASURED COMPLEXITY OF SOO. ALL FIGURES ARE WALL CLOCK TIMES EXPRESSED IN SECONDS. $T0 = 0.05$ SECOND.

Dimension	10	30	50	100
$T1$	0.06	0.26	0.67	1.93
$\hat{T2}$	1.14	1.70	1.92	3.38
$\frac{\hat{T2}-T1}{T0}$	21.6	28.8	25.0	29.0

C. Results

Regarding notations, for an objective function f, we note $f(x^*)$ the value of the best point found by SOO, and f^* the value of the optimum. All results are expressed as "function error value", that is $f(x^*) - f^*$.

1) Running time: All experiments were developed and executed on a Lenovo Thinkpad X220 based on an Intel Core i7-2640M CPU, 2.80GHz, 8Gb of main memory. The computer runs Ubuntu 13.10. All software is compiled with `gcc` version 4.8.1 using aggressive code optimization options.

We assess the complexity of SOO according to the challenge protocol [8]. Figures are given in table I. Function 18 of the challenge is considered. $T1$ is the computation time of 200000 objective function evaluations. Allowing 200000 function evaluations whatever the dimension, the running time (wall clock) averaged over 5 executions of SOO is denoted $\hat{T2}$. Hence, $\frac{\hat{T2}-T1}{T0}$ somehow quantifies the computational cost of running SOO itself.

2) Dependence on dimension: We run SOO on all 30 functions defined in 10, 30, 50, and 100 dimensions. We use the same number of function evaluations regardless of the dimension of the problem (which is not the competition protocol). The results are given in table II. Obviously, we

TABLE II

FOR EACH FUNCTION OF THE CEC'2014 COMPETITION DEFINED IN DIMENSIONS 10, 30, 50, AND 100, THIS TABLE GIVES THE VALUE OF THE BEST POINT FOUND BY SOO. IN EACH DIMENSION, THE NUMBER OF OBJECTIVE FUNCTION EVALUATIONS IS 10^5 WHATEVER THE DIMENSION OF THE DOMAIN: PLEASE NOTE THAT THIS IS NOT CEC'2014 COMPETITION SETUP.

Function	10 D	30 D	50 D	100 D
1	8.8×10^6	2.3×10^8	5.7×10^7	2.8×10^8
2	6.343	64377.6	8.2×10^7	1.5×10^9
3	6643.670	10810.8	12779.4	71342.3
4	0.678	111.129	351.202	1660.46
5	20.0	20.003	20.035	20.946
6	0.002	2.701	28.093	79.058
7	0.049	1.081	2.338	21.219
8	18.904	93.534	161.333	395.02
9	8.955	59.713	157.42	652.07
10	130.39	2608.24	4895.68	13104.2
11	349.05	2318.13	3945.9	13139.5
12	0.0	0.04	0.2	1.0
13	0.03	0.4	0.6	0.65
14	0.13	0.3	0.83	0.2
15	0.44	23.76	282.67	5521.64
16	2.52	10.81	19.4	42.08
17	3.1×10^6	2.8×10^7	1.9×10^8	1.6×10^8
18	12932.10	2943.96	47094.7	6.7×10^7
19	0.550	183.63	84.56	384.63
20	9364.20	38150.3	1.1×10^5	1.1×10^5
21	24694.90	1.6×10^7	5.0×10^7	1.2×0^8
22	126.460	1020.22	1639.28	3270.54
23	200.0	200.0	200.0	200.0
24	115.65	200.0	200.0	200.0
25	145.16	200.0	200.0	200.0
26	100.05	200.0	200.0	200.0
27	200.0	200.0	200.0	200.0
28	200.0	200.0	200.0	200.0
29	200.0	200.0	200.0	200.0
30	200.0	200.0	200.0	200.0

observe a degradation of results when the dimension increases. However, for a very significant subset of functions, the degradation is far from being severe. Indeed, for functions 5, 12, 13, 14, 23, 28, 29, and 30, the degradation between 10 D and 100 D is as little as 0.5%; moreover, for functions 7, 16, 24, 25, 26, and 27, the best found point is still very close to the optimum, their value being less than 10% larger that the optimum. It is also true that the performance is extremely poor on some of the objective functions, such as functions 1 and 2, and to a lesser extent on functions 3, 17, and 18. Functions 1 and 2 are difficult to optimize because the optimum is located in a very narrow region. Note that functions 17 and 18 combines 3 various functions, among which function 1 for 17, and function 2 for 18.

Following CEC'2014 competition protocol in which the number of function evaluations scales with the dimension ($10^4\times$ dimension of the problem), the results after the last function evaluation are given in table III.

By a virtue of using 3 times more function evaluations in 30D (resp. 5 times more in 50D), there is an average improvement of 9% (resp. 12%) of the minimum, the standard deviation of their improvement being 20% (resp. 21%). The median is 0.3% which reveals that most improvements are small (in 50D, the median of the improvement is larger: 6%).

TABLE III

FOR EACH FUNCTION OF THE CEC'2014 COMPETITION DEFINED IN DIMENSIONS 10, 30, 50, AND 100, THIS TABLE GIVES VALUE OF THE BEST POINT FOUND BY SOO. IN EACH DIMENSION, THE NUMBER OF OBJECTIVE FUNCTION EVALUATIONS IS $10^4\times$DIMENSION OF THE PROBLEM. THIS IS CEC'2014 COMPETITION SETUP. THE RESULTS IN 10 DIMENSIONS ARE THE SAME AS IN TABLE II THOUGH EXTRA PRECISION IS GIVEN HERE; WE MENTION THEM IN THE TABLE TO EASE COMPARISONS.

Function	10 D	30 D	50 D	100 D
1	8.8×1^6	2.2×10^8	5.3×10^7	2.1×10^8
2	6.343	31387	5.6×10^7	5.5×10^8
3	6643.670	10810	12152.1	55662.8
4	0.678	109.346	283.718	893.65
5	20.0	20.0	20.001	20.75
6	0.002	1.897	23.064	60.55
7	0.049	0.996	1.943	11.09
8	18.904	92.531	161.091	296.95
9	8.955	59.706	144.31	361.39
10	130.39	2312.38	4459.67	8612.37
11	349.05	2151.25	3924.15	9724.4
12	0.0	0.03	0.07	0.29
13	0.03	0.35	0.51	0.53
14	0.13	0.29	0.78	0.15
15	0.44	22.51	127.49	128.51
16	2.52	9.86	18.98	38.73
17	3.1×10^6	2.8×10^7	1.9×10^8	1.5×10^8
18	12932.1	2854.99	22655.0	1.3×10^6
19	0.55	183.62	82.48	339.1
20	9364.20	38149.6	1.1×10^5	94458.4
21	24694.9	1.6×10^7	5.0×10^7	9.3×10^7
22	126.46	1019.94	1628.97	2363.24
23	200.0	200.0	200.0	200.0
24	115.65	200.0	200.0	200.0
25	145.16	200.0	200.0	200.0
26	100.05	200.0	200.0	200.0
27	200.0	200.0	200.0	200.0
28	200.0	200.0	200.0	200.0
29	200.0	200.0	200.0	200.0
30	200.0	200.0	200.0	200.0

3) Dependence on the number of function evaluations: Again, we use 10^5 function evaluations regardless of the dimension.

In 10 dimensions, on functions 5, 12, 13, 14, 15, 16, 19, 23, 26, 28, 29, and 30, there is almost no progress between 10^4 and 10^5 function evaluations.

Figure 2 provides a graphical illustration of the improvement along function evaluations.

4) Comparison with DiRect: We use the implementation of DiRect available in the NLopt library [5] version 2.4.1. Due to lack of time, we compared DiRect and SOO only on problems in 10 dimensions. We perform 10^5 function evaluations at each run.

SOO outperforms DiRect on 17 functions and provides the same optimum on 6 other functions (*cf.* table IV). In our opinion, this is a strong case for SOO which enjoys formal properties that DiRect does not. Not mentioning that the SOO runs much faster than DiRect[1].

[1]though this might be implementaiton dependant as we use the implementation of DiRect available in the NLopt library, which is of high quality.

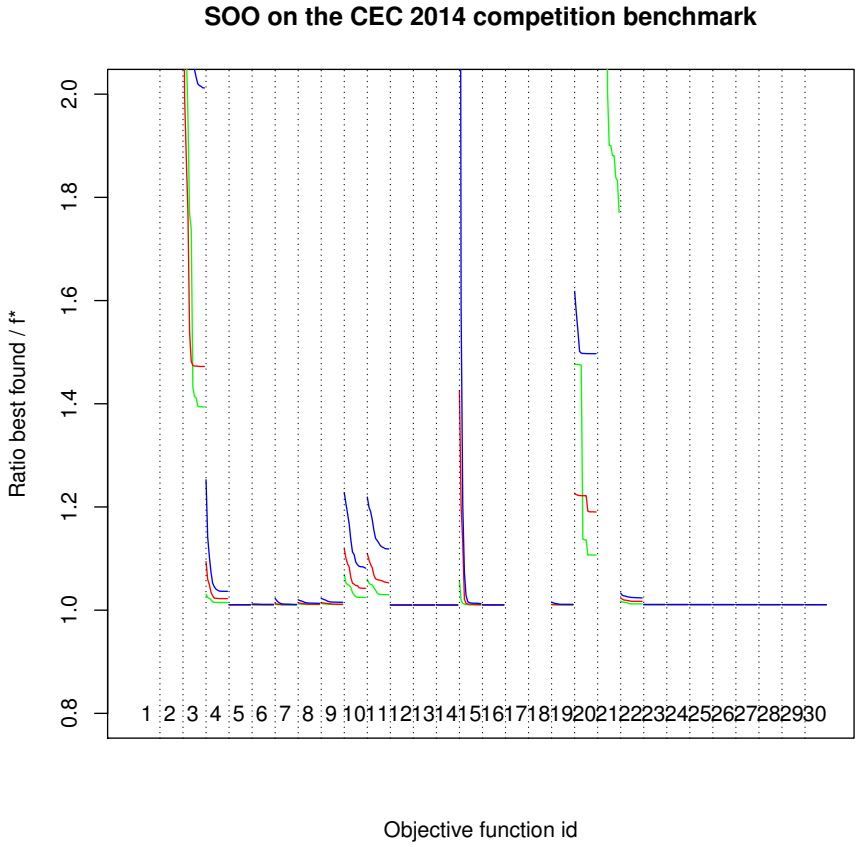

Fig. 2. For each of the 30 functions of the competition, we plot the ratio between the best point found and the optimum value along the optimization. We restrict the illustration to ratios that are ≤ 2. Colors indicate the dimension of the problem: black is 10D, green is 30D, red is 50D, and blue is 100D. Naturally, the larger the dimension, the larger the ratio in general.

5) Post-processing with a local optimizer: As seen above, increasing the amount of function evaluations does improve the result. However, the computational effort seems a little bit too much with regards to the gain. Actually, SOO does not provide a local optimum, only the best point it has found. Obviously, the results may be post-processed by some derivative free local optimizer. To respect the CEC'2014 competition setting, we did not consider gradient descent algorithms which requires the gradient of the objective function. We used derivative-free local optimizers, namely BOBYQA [11] and SUBLPX [13], as available in the already mentioned NLopt library.

The local optimizer also evaluates the objective function so that a trade-off between the number of evaluations by SOO and the local optimizer has to be found. There is also the issue of which point to locally optimize: indeed, SOO provides a set of points that may be optimized, not just one.

We have not yet studied thoroughly these issues, neither experimentally, nor theoretically. So, we assign 5% of the function evaluation budget to the local optimization of the best point found by SOO.

Table V gives the results on the 30 functions in dimension 10 and 30, using the BOBYQA optimizer (results obtained with SBPLX are not as good as those obtained with BOBYQA).

In 10 dimensions, the results are now very close to the optimum for all 30 functions. In 30 dimensions, a large majority of functions are optimized very close to their optimum.

This improvement was expected; it witnesses the fact that SOO does a very good job at identifying an area in which optimal or almost optimal points lay. Getting to the optimum is then only a matter of local optimization.

V. Conclusion and future work

In this paper, we have introduced the idea that the optimization of a function available through a black-box may be formulated as a problem of sequential decision making under uncertainty. Function optimization is thus a bandit problem, the aim being to find the best action to do after a fixed known amount of function evaluations. By so doing, we inherit the rich theory of bandits. This leads us to introduce the algorithm SOO. SOO is designed in order to meet certain formal properties regarding its performance after a finite amount of function evaluations. In this paper, we showed that SOO is

TABLE V

FOR EACH FUNCTION OF THE CEC'2014 COMPETITION DEFINED IN DIMENSIONS 10, AND 30, , THIS TABLE GIVES VALUE OF THE BEST POINT FOUND AFTER POST-PROCESSING THE BEST POINT FOUND BY SOO WITH BOBYQA. IN EACH DIMENSION, THE NUMBER OF OBJECTIVE FUNCTION EVALUATIONS IS $10^4 \times$DIMENSION OF THE PROBLEM. THIS IS CEC'2014 COMPETITION SETUP. THE RESULTS IN THE FIRST AND THIRD COLUMNS ARE THE SAME AS IN TABLE III; WE MENTION THEM IN THE TABLE TO EASE COMPARISONS.

Function	10D		30D	
	no post-optimization	post-optimization	no post-optimization	post-optimization
1	8.8×10^6	4569.72	2.2×10^8	2674850.0
2	6.343	0.04	31387.0	99.61
3	6643.67	5842.92	10810	7840.39
4	0.678	0.0	109.346	36.75
5	20.00	20.0	20.0	20.0
6	0.002	0.00	1.897	1.91
7	0.049	0.05	0.996	0.41
8	18.904	18.90	92.531	92.53
9	8.955	8.96	59.706	59.7
10	130.39	130.39	2312.38	2131.47
11	349.05	349.05	2151.25	2091.05
12	0.0	0.0	0.03	0.03
13	0.03	0.03	0.35	0.34
14	0.13	0.13	0.29	0.28
15	0.44	0.42	22.51	21.69
16	2.52	2.52	09.86	09.81
17	3.1×10^6	322.57	2.8×10^7	42148.7
18	12932.1	3951.62	2854.99	41.58
19	0.55	0.55	183.62	16.3
20	9364.2	6925.1	38149.6	34381.2
21	24694.9	1940.39	1.6×10^7	15435.0
22	126.46	126.47	1019.94	956.48
23	200.0	200.0	200.0	200.0
24	115.65	115.65	200.0	200.0
25	145.16	139.08	200.0	200.0
26	100.05	100.05	200.0	200.0
27	200.0	200.0	200.0	200.0
28	200.0	200.0	200.0	200.0
29	200.0	200.0	200.0	200.0
30	200.0	200.0	200.0	200.0

an interesting global optimization algorithm. We provide an experimental study of SOO on the CEC'2014 competition on single objective real parameter numerical optimization. Though probably not the best global optimizer available today, we think that SOO has demonstrated its interest as a global optimization algorithm. In particular, against common intuition, and against our own expectation, SOO is able to perform remarkably well in high dimensional functions, the degradation of performance between 10D and 100D being sometimes very small. SOO is a very simple algorithm, hence very simple to implement, and rather fast. Moreover, SOO may be combined with a local optimization algorithm and consequently providing greatly improved results.

After this work, there are many questions that we are going to study further. 1) Regarding parameter tuning, we have not done an extensive study, and we think that some parameters may take advantage of an online auto-tuning. 2) Due to the lack of time, we have not tried to compete on the part B of the competition in which the amount of function evaluations is very small, but one may learn a model to find a better optimum, or even guide the search. There are very interesting theoretical questions behind that. 3) Being a global optimizer, SOO is not at ease with local optimization in high dimensions; so, it makes sense to post-process the best point(s) found by SOO by some local optimization procedure. We did some experiments using this idea, though a more complete experimental study is due; furthermore, again, there are very interesting questions to investigate from a theoretical point of view, in particular the trade-off between SOO and the local optimizer[2]. 4) The characterization of a cell by the point in its center may also be a path of future work. Restricting the behavior of the objective function by its

[2]As a matter of fact, we have recently learned about György and Kocsis paper [4] which addresses this issue.

TABLE IV

COMPARISON OF THE RESULTS OBTAINED WITH SOO AND DIRECT ON THE 30 FUNCTIONS IN 10 DIMENSIONS. ON EACH FUNCTION, EACH ALGORITHM PERFORMS 10^5 EVALUATIONS. BOLD FACE INDICATES THE BEST RESULT FOR EACH FUNCTION.

Function	DiRect	SOO
1	$\mathbf{7.5 \times 10^6}$	8.8×10^6
2	514.399	**6.343**
3	**6332.04**	6643.67
4	**4.284**	0.678
5	0.0	0.0
6	4.27	**0.002**
7	0.487	**0.049**
8	31.839	**18.904**
9	30.844	**08.955**
10	604.17	**130.39**
11	1549.37	**349.050**
12	0.31	**0.0**
13	0.19	**0.03**
14	0.17	**0.13**
15	1.91	**0.44**
16	3.1	**2.52**
17	$\mathbf{5.6 \times 10^5}$	3.1×10^6
18	**12810.6**	12932.1
19	3.95	**0.550**
20	**9082.8**	9364.2
21	**24247.6**	24694.9
22	441.2	**126.46**
23	200.0	200.0
24	133.94	**115.65**
25	200	**145.16**
26	200.34	**100.05**
27	200.0	200.0
28	200.0	200.0
29	200.0	200.0
30	200.0	200.0

value on a single point of a cell, and particularly its center, is really a choice that should be challenged. We may consider sampling different points of the cell to determine some sort of average behavior in the cell and an associated higher order moments. From this, we would be able to derive a confidence bound regarding each cell that would guide the choice of the next cell to split. This links this work with its original motivation of optimizing nondeterministic functions. Indeed, SOO is one member of a family of algorithms, all based on the same principles. Other algorithms assume certain properties on the objective function (HOO and DOO assume that the local smoothness of the function is known and is required to tune the algorithm); StoSOO [15] is an extension of SOO which performs function optimization of noisy (or stochastic) functions. The interested reader may also find the survey of R. Munos [10] useful. All these algorithms may optimize functions having various properties; the principles of UCB and of optimism in face of uncertainty may lead to other algorithms based on the assumptions made about the objective function.

REFERENCES

[1] J.-Y. Audibert, S. Bubeck, and R. Munos. Best arm identification in multi-armed bandits. In *Conference on Learning Theory*, 2010.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47((2-3)):235—256, 2002.

[3] S. Bubeck, R. Munos, and G. Stoltz. Pure exploration in multi-armed bandits problems. In *Proc. of the 20th International Conference on Algorithmic Learning Theory*, pages 23–37, 2009.

[4] A. György and L. Kocsis. Efficient multi-start strategies for local search algorithms. *Journal of Artificial Intelligence Reseach (JAIR)*, 41:407–444, 2011.

[5] Steven G. Johnson. The NLopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`.

[6] D.R. Jones, C.D. Perttunen, and B.E. Stuckman. Lipschitzian optimization without the lipschitz constant. *J'al of Opt. Theory and Applications*, 79(1):157–181, 1993.

[7] T.L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4—22, 1985.

[8] J.J. Liang, B.Y. Qu, and P.N. Suganthan. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. Technical Report 201311, Nanyang Technological University, Singapore, December 2014. `http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2014/CEC2014.htm`.

[9] R. Munos. Optimistic optimization of deterministic functions without the knowledge of its smoothness. In *Advances in Neural Information Processing Systems*, 2011.

[10] R. Munos. From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, pages 1–130, 2014.

[11] M.J.D. Powell. The bobyqa algorithm for bound constrained optimization without derivatives. Technical Report technical report NA2009/06, Department of Applied Mathematics and Theoretical Physics, 2009.

[12] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematics Society*, 58:527—535, 1952.

[13] T. Rowan. *Functional Stability Analysis of Numerical Algorithms*. PhD thesis, Department of Computer Sciences, University of Texas at Austin, 1990.

[14] W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25:285—294, 1933.

[15] Michal Valko, Alexandra Carpentier, and Rémi Munos. Stochastic simultaneous optimistic optimization. In *International Conference on Machine Learning*, 2013.