

Optimal robust expensive optimization is tractable

P. Rolet, M. Sebag, O. Teytaud,
Tao, Inria, Lri, Umr Cnrs 8623, Univ. Paris-Sud

ABSTRACT

Following a number of recent papers investigating the possibility of optimal comparison-based optimization algorithms for a given distribution of probability on fitness functions, we (i) discuss the comparison-based constraints (ii) choose a setting in which theoretical tight bounds are known (iii) develop a careful implementation using billiard algorithms, Upper Confidence trees and (iv) experimentally test the tractability of the approach. The results, on still very simple cases, show that the approach, yet still preliminary, could be tested successfully until dimension 10 and horizon 50 iterations within a few hours on a standard computer, with convergence rate far better than the best algorithms.

Categories and Subject Descriptors

G.1.6 [Optimization]: Nonlinear programming; I.2.8 [Problem Solving, Control Methods, and Search]: Graph and tree search strategies

General Terms

Algorithms

1. INTRODUCTION

Computational optimisation techniques are concerned with a growing variety of fields, and among them those implying expensive-to-evaluate fitnesses. Industries such as aerospace or automobile often rely on numerical engineering: the codes used for simulations during optimisation processes are computationally heavy. In the web industry, many web applications try to learn people's tastes by using preference ranking techniques that present a user choices (usually binary): the fitness is the user's utility function, and each iteration is costly since the user has to think about his tastes before answering; furthermore, too many requests could annoy him/her.

This paper is interested in optimal performance of optimisation problems given a small number of allowed iterations

(i.e. fitness evaluations) that characterize expensive optimization. Besides, focus will be set on *robust* optimisation—i.e. rugged fitness functions—and *comparison-based* algorithms such as in evolutionary computation.

Robust comparison-based optimisation

Evolutionary algorithms are often said to be of order 0, because they don't rely on gradient computation. It is however possible to distinguish between two kinds of "order 0" algorithms: those using fitness evaluations and those only requiring comparisons between fitness values, that is binary informations. Algorithms based on surrogate models (see for instance [25]) usually fall in the former category (with the exception of [19]). However, most evolutionary algorithms are comparison-based. Comparison-based strategies have nice robustness guarantees, and there are known complexity bounds relating precision and number of iterations, which is particularly needed in an expensive optimization setting. Besides, in some applications—for instance in preference ranking problems introduced above—comparisons are the only available information. Therefore, much work has already been done regarding these strategies.

Expensive optimisation

Many artificial fitness functions are computed very quickly, whereas real-world applications often involve huge computational cost. This can be due, for instance, to finite-element methods or (Quasi-)Monte-Carlo sampling in numerical engineering problems. In some cases the fitness function can take days to be evaluated on a point. In that case, one can often neglect the internal cost of the optimization algorithm, and only consider the number of fitness evaluations—the fact that optimization algorithms may require a few minutes before sending a request to the fitness function does not matter. Therefore, some algorithms with a huge internal computational cost have been designed. Efficient Global Optimization [15], and Informational Approach to Global Optimization [22, 23] are examples of such algorithms. They are robust in front of local minima, and experimentally quite good; on the other hand, as acknowledged by the authors of [23], their algorithm is untractable in high dimension due to the selection among a finite set of candidates (with size exponential in the size of the domain), and has no convergence proof; also, it is based on a Gaussian prior on the distribution of fitness functions (i.e. the fitness functions is supposed to be drawn according to a Gaussian process).

In this paper we use an approach similar to [5]: a novel way to reach optimal optimisation in the expensive setting

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

based on Monte-Carlo tree search, along with techniques allowing this approach to be tractable. This includes (i) proved optimality under some arbitrary prior (ii) optimized implementation through billiards, progressive widening and other techniques (iii) experiments in settings in which the optimal possible convergence rates are known, leading to a proof of concept.

In section 2.1, we recall results from [11] showing that for a general robustness criterion introduced below, optimality can be reached without using more informations than comparisons (i.e. there are optimal optimization algorithms which are comparison-based).

In section 2.2, we present state-of-the-art complexity bounds for such algorithms [21, 20].

In section 3.2 and 3.3, we introduce Upper Confidence Trees (UCT)[16], a Monte-Carlo tree search algorithm, and show how it can be used for approximating an optimal comparison-based optimization algorithm. This has already been tried in [5] with very small scale results; we will therefore show in section 4 how to drastically reduce the computational cost.

Section 5 will detail experiments on the resulting algorithm showing how it gets close to complexity bounds from [21, 20] introduced in section 2.2.

2. THEORETICAL BACKGROUND

Section 2.1 recalls theoretical results showing that comparison-based optimization is optimal with respect to a standard robustness criterion. Then, section 2.2 shows that the comparison-based nature of an optimization algorithm entails complexity bounds: it is not possible to be faster than some absolute limit. Next sections will show that this absolute limit can be reached theoretically (by an algorithm with huge computational cost), and that it can be approximated thanks to the UCT algorithm.

2.1 Optimal optimization with robustness constraints

We consider families of optimization algorithms. Thus we need to formalize optimization algorithms. We consider a real-valued fitness function f defined on D .

For some fixed optimization algorithm Opt generating a point x_N as an approximation of the optimum of the fitness function f after N steps, the performance is evaluated e.g. by $\|x_N - \arg \min f\|^2$, f being assumed to have one global minimum. The optimization algorithm is formalized as a map $Opt: \{\emptyset\} \cup \bigcup_{n=1}^N (D \times \mathbb{R})^n \rightarrow D$ and we define the sequence $Opt_f = (x_n)_{1 \leq n \leq N}$, as follows:

$$x_1 = Opt() \text{ and}$$

$$\forall n \in \{1, \dots, N-1\}, x_{n+1} = Opt(x_1, f(x_1), \dots, x_n, f(x_n)).$$

We introduce the set of increasing functions $G = \{g : \mathbb{R} \rightarrow \mathbb{R}; \forall (x, y) \in \mathbb{R}^2, x < y \Rightarrow g(x) < g(y)\}$. An algorithm $f \mapsto Opt_f$ is said *comparison-based* if, for every $g \in G$, the output of the map Opt is the same if the values y_1, \dots, y_n in its inputs are replaced by $g(y_1), \dots, g(y_n)$, that is, for all $x_1, \dots, x_n \in D, y_1, \dots, y_n \in \mathbb{R}$ and $g \in G$, $Opt(x_1, g(y_1), \dots, x_n, g(y_n)) = Opt(x_1, y_1, \dots, x_n, y_n)$.

We consider *robustness properties*. In the robust case, the quality of an optimizer is estimated by its worst case among a family of functions: if F is a space of fitness functions, N is a number of iterations and x_N is the estimate of the

optimum proposed by the algorithm, the quality criterion is $\sup_{f \in F} \|x_N - x^*(f)\|^2$ where $x^*(f)$ is the optimum of the fitness function f . For this robustness criterion, theorem 2.1 shows that if F is stable by composition with G , then every optimization algorithm can be replaced without loss of efficiency by a comparison-based algorithm.

Many algorithms, in spite of this restriction (they only use comparisons, and loose all other information), have been proved to be linear in the sense that the log-distance to the optimum converges to $-\infty$ linearly in the number of iterations (i.e. $\frac{1}{n} \log \|x_n - x^*(f)\|$ converges to a negative constant); see e.g. [3, 4, 9, 18]. Therefore these algorithms have a reasonably good convergence rate. Some linear lower bounds also exist in various cases [14, 21], and they show that the constant in the linear convergence decreases to 0 linearly with the inverse of the dimension.

Theorem 2.1 below states that, for this robustness criterion, for every optimization algorithm Opt , there is another optimization algorithm Opt' that has the same efficiency and such that Opt' is comparison-based. More precisely, we state that if for some N and ϵ , Opt ensures that the N -th iteration is the optimum within precision ϵ , then there exists Opt' which is comparison-based and ensures the same precision.

We let $sign(x) = 1$ if $x > 0$, $sign(x) = -1$ if $x < 0$ and $sign(0) = 0$.

THEOREM 2.1 (COMPARISONS ARE ROBUST).

Consider F a space of real-valued functions defined on a given domain D such that each $f \in F$ has one and only one global minimum, and assume that, for all $f \in F$ and all $g \in G$, $g \circ f$ belongs to F . Consider a deterministic optimization algorithm $Opt : f \mapsto Opt_f = (x_n)_{n \in \mathbb{N}}$ (see the definitions in Section 1). We consider $x'_N = (Opt_f)_N$ as a function of f . Assume that, for some $\epsilon > 0$, there exists an integer N such that,

$$\forall f \in F, \|(Opt_f)_N - \arg \min f\| \leq \epsilon. \quad (1)$$

Then, there exists a deterministic algorithm Opt' that only depends on comparisons, in the sense that

$$\begin{aligned} &(\forall i, j, sign(y_i - y_j) = sign(y'_i - y'_j)) \\ \implies &Opt'(x_1, y_1, \dots, x_n, y_n) = Opt'(x_1, y'_1, \dots, x_n, y'_n), \end{aligned}$$

and Opt' is such that Equation (1) holds with the same ϵ and the same N , i.e.,

$$\forall f \in F, \|(Opt'_f)_N - \arg \min f\| \leq \epsilon. \quad (2)$$

The reader is referred to [11] for the proof and variations of these results in terms of convergence rate.

2.2 Consequences in terms of complexity

[21] (extended in [20]) has shown that comparison-based algorithms have some ultimate limits on their convergence rates. Results in [21, 20] are much more general than that, but we will restrict here our attention to the case of algorithms using one single binary information, i.e. one comparison between two fitness values. Theorem 1 in [21] implies that, if x_n is proposed by a comparison-based algorithm as an approximation of the optimum after n comparisons, then necessarily, for the worst case among objective functions in any translation invariant family F of fitness functions on $[0, 1]^d$ with one and only one optimum, the expected dis-

tance between x_n and the optimum x^* is as follows:

$$\lim_{n \rightarrow \infty} \sup_{f \in F} \mathbb{E} \frac{d}{n} \log_2(\|x_n - x^*(f)\|) \geq -1. \quad (3)$$

3. OPTIMAL COMPARISON-BASED ALGORITHMS

[11] provides, as a preliminary approximation of optimal optimization algorithm, BREDA. BREDA uses as prior information a space F of fitness functions; f is supposed to be distributed according to the probability distribution \mathcal{F} .

Then, BREDA uniformly draws an offspring in the set S of points which might be an optimum, i.e.:

$S =$

$\{\arg \min f; f \in F, f \text{ is consistent with previous observations}\}.$

If previous observations are comparisons, i.e. in previous offsprings it has been established that

$$\begin{aligned} f(x_1) &\leq f(x'_1) \\ f(x_2) &\leq f(x'_2) \\ f(x_3) &\leq f(x'_3) \\ &\dots \\ f(x_n) &\leq f(x'_n) \end{aligned}$$

then this means

$$S = \{\arg \min f; f \in F, \forall i \in \{1, 2, \dots, n\} f(x_i) \geq f(x'_i)\}.$$

This random draw is done using a *billiard* algorithm, the principle of which is described in section 4. In order to do so, we need a family of possible fitness functions, and the memory of past comparisons in order to reduce the set of possible fitness functions; S is then the set of the optima of all the fitness functions which are consistent with previous comparisons. BREDA has very good empirical results.

However, BREDA is not rigorously optimal. Here, we rephrase the optimal optimization problem as a Partially Observable Markov Decision Process (POMDP):

- A decision consists in choosing the next points for which fitnesses are compared.
- Randomness consists in choosing the fitness function (once for all; this is the unobservable part of the MDP).
- The observations are the points which are compared (i.e. the decisions) and the results of these comparisons.
- The horizon (number of time steps) is the budget in terms of requests for comparisons.

[1] has shown that POMDP can be solved as standard Markov Decision Processes (MDP), provided that the state is augmented so that it contains all the past observations. [5] uses this in order to rephrase the POMDP of comparison-based optimal non-linear optimization as a MDP, and then approximately solves this MDP in a very restricted setting. This section is devoted to

- present UCT, an algorithm which is particularly suitable for this problem (section 3.2);
- show its applicability in our framework, namely optimal comparison-based algorithm (section 3.3).

3.1 Upper Confidence Bounds in a nutshell

Imagine you are in a node of a graph, and have to choose between various edges e_1, e_2, \dots, e_k . Also, assume you have, in the past, already tested these edges:

- you tested n_1 times edge e_1 , with on average a reward r_1 (defined if and only if $n_1 > 0$);
- you tested n_2 times edge e_2 , with average reward r_2 (idem);
- ...
- you tested n_k times edge e_k , with average reward r_k (idem).

Then, the Upper Confidence Bound (UCB, [17, 2]) score of edge e_i is

$$r_i + C \sqrt{\log(\sum_j n_j) / n_i} \quad (4)$$

if $n_i > 0$, and ∞ otherwise, where C is some empirically tuned constant. Then, the UCB algorithm suggests to choose edge e_i such that s_i is maximal, in order to get fruitful information. An important point is that e_i is not necessarily the edge which is expected to provide the best reward; it is supposed to be a good choice in order to get more information for the future. UCB is the guide for choosing paths to be explored in the tree of the MDP by the Upper Confidence Trees algorithm, to be presented below.

3.2 Upper Confidence Trees in a nutshell

Upper Confidence Trees (UCT, [16]) is a particular form of Monte-Carlo Tree Search [7, 6] based on multi-armed bandit formalism and the UCB formula [2]. The algorithm has been applied to the game of Go with impressive results [24, 12, 10] (first victories of a computer against professional human). In spite of the fact that it is mainly known in the artificial game players community, it can be applied in numerous other similar settings, such as in discrete time control problems, even in the stochastic case.

A detailed presentation of UCT is beyond the scope of this paper. Therefore, the reader is referred to [16] for a more detailed presentation; we only here informally introduce UCT.

Remember the goal is to select the right actions in the graph of the MDP at each step, so that the reward is maximised in expectancy when the horizon T is reached. Thus, a node of the graph can be viewed as a multi-armed bandit, with each branch having an unknown mean reward, and branches that look more rewarding should be searched more deeply—ultimately only looking at the actual most rewarding branch. There is an exploration/exploitation dilemma lying in the uncertainty in current estimates of branches' mean reward. A property of the UCB formula is that asymptotically, the branch that has the best mean reward is played exponentially more times than the others[2]. When a leaf is reached, a reward is given and estimates of mean reward for the nodes of the currently explored path are updated accordingly.

UCT is a Monte-Carlo strategy: it relies on performing many random walks in the graph. It typically proceeds this way: run a random simulation – i.e randomly (but not uniformly!) select actions to go from s_0 (the initial state) to a

terminal state s_T – and get a reward. This reward is seen as a first estimate of the states’ value. Perform random simulations again, until all actions at the root s_0 have been tried once. From this moment on, use the UCB formula to select the action to take at s_0 for each new simulation, and then proceed in the same way with child states of s_0 . As the number of simulations increases, more and more deep states are being explored as multi-armed bandits, and the estimates of expected rewards for the actions converge to their true values. One can then select the best unlabeled sample from s_0 , which is an estimate of what the optimal strategy would recommend, and label it with the real hypothesis, the one that must actually be learnt. Eventually, the approximated policy will converge to the optimal policy as the number of simulations tend to infinity.

This outlines two important properties of UCT:

- Asymmetric tree search: the tree is developed more precisely on better looking branches in a dynamic fashion;
- Anytime: The result will improve somewhat continuously with the number of random walks that are made, as opposed to e.g. minimax tree search; stopping search at any time may yield a significant result;

3.3 UCT in the optimization framework

As discussed in the beginning of section 3, optimal comparison-based optimization can be rephrased as a POMDP. Thanks to [1], this MDP can be rephrased as a MDP. We have presented above UCT for solving such a MDP. Let us now formalize this MDP so that the complete algorithm becomes clear.

\mathcal{X} is the space of possible requests and \mathcal{Y} is the space of possible answers to these requests; typically, \mathcal{X} might be D^2 , where D is the domain of the optimization, and \mathcal{Y} might be $\{0, 1\}$. An answer $y = Oracle_f((x, x'))$ to a request $(x, x') \in \mathcal{X} = D^2$ is then 1 if $f(x) \leq f(x')$, and 0 otherwise:

$$Oracle_f(x, x') = 1 \text{ if and only if } f(x) \leq f(x') \\ = 0 \text{ otherwise} \quad (5)$$

We restrict the discussion below and our implementation to this case, but we could also consider λ points (x_1, \dots, x_λ) as a request, and \mathcal{Y} the set of orders (with possibly equality) among λ points.

In figure 3.3 is a tree¹ representing the MDP. Let us remind the correspondence in vocabulary between optimization and MDPs. *Decision* nodes are labelled with a state, i.e. a finite sequence $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ in $\mathcal{X} \times \mathcal{Y}$; from decision nodes, we have edges (an edge is labelled by some unanswered request x) directed to *random* nodes. A random node is labelled with a finite sequence $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ in $\mathcal{X} \times \mathcal{Y}$, plus the unanswered request x ; there is an edge labelled by decision $x \in \mathcal{X}$ from the decision node labelled by $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ to the random node labelled by $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ and x . There are two edges, labelled by 0 and 1, from this random node, leading to nodes

¹It is actually a directed acyclic graph, since for instance state $s_2 = \{(x_1, y_1), (x_2, y_2)\}$ can be reached either from state $s_1 = \{(x_1, y_1)\}$ or $s_2 = \{(x_2, y_2)\}$. However, as cycles are rare, the usual terminology keeps the word "tree"

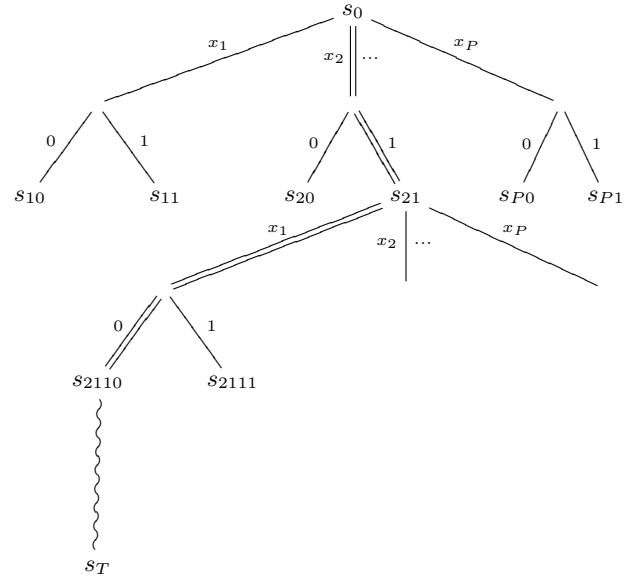


Figure 1: A tree representation of the MDP. Once an action x is chosen (i.e. a pair of points to be compared) at state s_0 , the "label" stage of the tree correspond to transition function $p_{tr}(\cdot | s_t, x)$; it is the probability that a hypothesis randomly chosen in the set of hypotheses consistent with s_t labels x by 0 or 1.

labelled by respectively $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k), (x, 0)$ and $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k), (x, 1)$.

The probability on the edge from the node labelled $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ and x , to the node labelled by $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k), (x, 0)$ is $P(Oracle_f(x) = 0 | Oracle_f(x_1) = y_1, Oracle_f(x_2) = y_2, \dots, Oracle_f(x_k) = y_k)$.

4. IMPLEMENTATION AND PRACTICAL ISSUES

UCT has already been applied to optimization in [5]; however, it was not fully documented (the paper was essentially theoretical) and tested only in very specific cases. We here fully detail the implementation and experiment it on a much larger (yet still restricted) case. Moreover, we compare the results with theoretical bounds discussed in section 2.2 and proved in [21, 20].

We assume that a distribution of probability \mathcal{F} on the space F of fitness functions is given. The optimization algorithm is allowed to send requests about the unknown fitness function f ; the oracle send provides a label for this request. For example, the request is a population of points, and the oracle gives the comparisons between fitness values. In our experiments, we will consider two points x and x' , and the oracle answer 1 if and only if $f(x) \leq f(x')$, but the principle is generic. The resulting Algorithm 1 is termed CONO (comparison-based optimal non-linear optimization).

The first point is tackled in the following way. Before starting a new random walk from s_0 , a hypothesis (a fitness function) f is chosen randomly in \mathcal{F} . It will play the role of the target hypothesis for this walk. At a given state, when

Algorithm 1 The Cono algorithm. Given a measure on the set \mathcal{F} of fitness functions, a number of simulations N and a target hypothesis h^* , the main loop takes as argument a horizon T , a current state s (starting by \emptyset) and a current horizon t (starting by T). It runs N simulations as described above, and chooses the action x whose mean reward estimate is the best to label by h^* . It then calls itself after decrementing by 1 the horizon and restricting the hypothesis space to hypotheses consistent with $(x, h^*(x))$, runs N simulations again, and repeats the process until $T = 0$. An admissible request is a pair (x_1, x_2) such that for $i \in \{1, 2\}$, $x_i \in \arg \min h$ for some $f \in \mathcal{F}$ consistent with previous requests (i.e. we only allow the algorithm to compare points which might be the optimum - this is directly inspired by BREDA).

Function CONO(s, t)

if $t == 0$ then

 Return $\arg \min_{x \in D} \mathbb{E}_{f \in \mathcal{F}} \|x - \arg \min f\|^2$

end if

for $i=1$ to N do

 PerformSimulation(s, t)

end for

// find the request to be sent to the oracle

Let x be the action corresponding to the most visited branch from s

Let y be the label of x , i.e. $y = Oracle_f(x)$

Return CONO($s \cup \{(x, y)\}, t - 1$)

Function PerformSimulation(s, t)

Pick f using the distribution of probability \mathcal{F}

/** Oracle $_f$ is defined as in Eq. 5 **/

for $i = 1$ to t do

 Get the admissible branches, i.e.:

$n_{br} = PW(s)$

$P_{adm} = n_{br}$ requests randomly drawn among admissible requests.

 Get the best branch and label it by h , i.e. :

$x_0 = \arg \max_{x \in P_{adm}} UCB(s \cup \{x\})$

$s \leftarrow s \cup \{(x_0, Oracle_f(x_0))\}$

end for

Let $x = \arg \min_{x \in D} \mathbb{E}_{f \in \mathcal{F}} \|x - \arg \min f\|^2$

Get the reward $r = \|x - \arg \min f\|^2$

Update the tree by increasing the number of visits of each node on the path by one, and adjusting their mean rewards by r

Function UCB($s \cup \{x\}$)

Returns the UCB value of branch x taken as node s , or $+\infty$ if the branch has never been tried

Function PW(s) /** progressive widening **/

Return the number of branches that should be considered at node s , here $\lfloor n_s \rfloor^{1/4}$ where n_s is the number of times s has been visited

an action x is chosen, the next state is determined by the label $Oracle_f(x)$.

To deal with the second point, a technique called *progressive widening* [8] is applied. It basically limits the number of branches that are allowed to be explored from a state, based on the number of times that the state has been visited. In this implementation, a node that has been visited n times will explore only a subset of possible actions whose cardinal is $\lfloor n^{1/4} \rfloor^2$. For instance, for the first fifteen random walks, only one action will be explored from the root node; at the sixteenth simulation, another action will be tried and UCB will be applied on these two actions. Progressive widening also has the added benefit of removing the need for the finite pool assumption.

A pseudo-code of the algorithm, called Cono (for comparison-based optimal nonlinear optimization) is presented in figure 1.

Remarks on the algorithm 1.

- **Billiard algorithms.** We have to randomly draw two possible optima, to be compared by the oracle (i.e. to be ranked depending on their fitness values). This is performed in our implementation thanks to billiard algorithms, as in [13] and [5]. The algorithm is presented in Algorithm 2; informally the principle consists in launching a particle in \mathbb{R}^d , which bounces on satisfied constraints and crosses freely unsatisfied constraints. After a finite length, the particle halts; this is the output of the billiard algorithm, and this output is expected to be uniformly drawn in the points which satisfy the constraints, in the limit of an infinite length path. This property is only proved in very restricted cases and is conjectured in the general case.

Algorithm 2 Billiard algorithm, taking as input a set of constraints g_i , a trajectory length L , and returning a final point. The algorithm is supposed to return a final point uniformly drawn in $\{z \in \mathbb{R}^d; \forall i, g_i(z) \geq 0\}$, in the limit of $L \rightarrow \infty$.

Randomly select $z \in \mathbb{R}^d$ satisfying at least one constraint, and a direction \vec{v} .

while $L > 0$ **do**

 //Find the set of satisfied constraints

$J = \{j; g_j(p) \geq 0\}$

 //Go as far as possible while $g_j \geq 0, j \in J$

$\lambda^* = \sup\{\lambda \geq 0 \text{ s.t. } \forall \ell < \lambda, \forall j \in J, g_j(z + \ell \vec{v}) \geq 0\}$

if $J = \{1, \dots, n\}$ **then**

 //(all constraints satisfied)

if $L > \lambda^*$ **then**

 //Go until some g_i is saturated

$p = p + \lambda^* \vec{v}$

$L = L - \lambda^*$

else

 Return $p + L \vec{v}$

 //out of resources

end if

end if

$\vec{v} = \text{symmetric}(\vec{v}, g_i)$

 //Bounce against g_i

end while

- **About which request should be chosen after the simulations have been performed.** The action se-

²This formula is used on the basis of its empirical success (see [8, 6])

lected at the end of a UCT round (N simulations) is the action that has been the most visited—it is known as a robust choice, much more efficient than a criterion like best rewards.

The algorithm takes a number of simulations as input, but since the approach is anytime it would be easy to give it a time budget instead, if required.

5. EXPERIMENTS

This section compares (i) BREDA (ii) UCT built on top of BREDA (iii) theoretical bounds in the following cases:

- the domain D of the optimization is $[0, 1]^d$;
- the horizon is 10, 20, 50;
- the fitness function is the sphere function, i.e. a point x has better fitness than a point y for the sphere function centered at x^* if and only if $\|x - x^*\| < \|y - y^*\|$.

Results are presented in Fig. 2, 3, 4 respectively, for horizon 10, 20, 50.

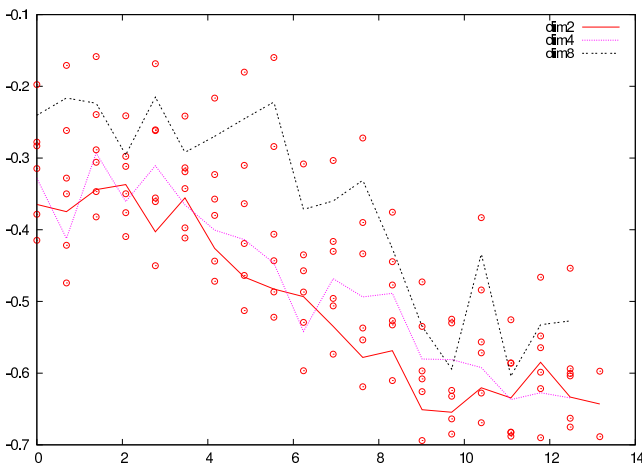


Figure 2: Results for horizon $n = 10$ with various dimensions. Error bars are $1/3$ of the standard deviation. The abscissa is the \log_2 of the number of simulations, the ordinate is the average of $\log(\|x_n - x^*(f)\|) \times d/n$, supposed to reach $-\log(2) = 0.69315$ asymptotically in n if bound in Eq. 3 is reached.

6. CONCLUSION

Comparison-based algorithms are optimally robust for some robustness criterion. This paper has shown the feasibility, thanks to UCT and to billiard algorithms, of optimal comparison-based algorithms; in particular, BREDA [11], which has provided the current best results on the sphere function, could be outperformed whilst preserving a reasonable computation time.

The point in this section is to estimate the tractability of the approach. The conclusions are as follows:

- Each curve in figure 2, 3, 4 is based on 12 hours of computation, for the whole curve, i.e. all tested numbers

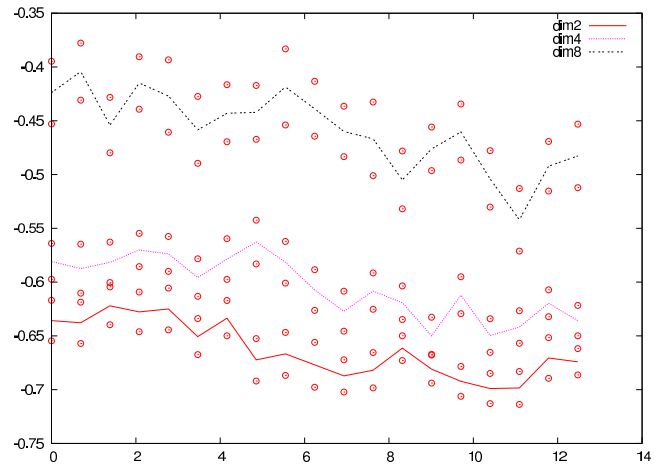


Figure 3: Results for horizon $n = 20$ with various dimensions. Error bars are $1/3$ of the standard deviation. The abscissa is the \log_2 of the number of simulations, the ordinate is the average of $\log(\|x_n - x^*(f)\|) \times d/n$, supposed to reach $-\log(2) = 0.69315$ asymptotically in n if bound in Eq. 3 is reached.

of simulations. The optimization runs are therefore at most a few hours long, accumulating to 12h for all experiments with 1,2,4,8,16,... simulations.

- The scores are close to optimality for horizon 10 in dimension 2, 4, 8; for horizon 20 in dimension 2 and 4; for horizon 50, in dimension 2 and 4.
- Even in non-optimal cases, the results were better than BREDA which is already much faster than most ES. BREDA is the case of 1 simulation and CONO successfully benefits from its UCT-based exploration for outperforming BREDA with more simulations. We recall that typical constants in most ES are 0.06 instead of 0.69 at the limit of CONO (almost reached in many cases, and almost always more than 0.5).

7. DISCUSSION

This paper is a part of a long-term work in progress. In [11], it was shown that comparison-based algorithms are optimal for some robustness criterion, and a theoretical algorithm was proposed. In [5], an approximation of optimal algorithm, using UCT, was outlined but was still incredibly slow. In [21, 20], complexity lower-bounds for comparison-based algorithms were derived. In this paper, some tricks (e.g. the use of the BREDA algorithm) have been added, so that the algorithm can be tested more intensively and in a more general setting; also, we have tackled problems for which optimal constants can be theoretically derived - we have by the way shown that optimality is very well approximated. A few hours were enough for results close to optimal constants, for up to 8 dimensions and up to 50 iterations. A cluster was used for constructing error bars, but each run was performed on a single core, within a few hours, each complete curve with 1, 2, 4, 8, ... simulations being generated in 12 hours.

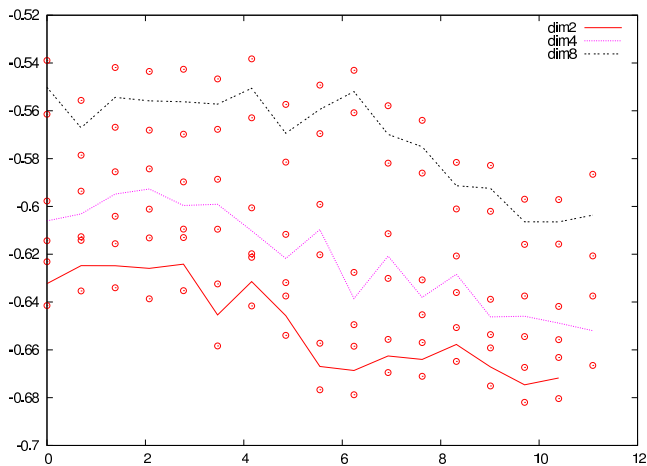


Figure 4: Results for horizon $n = 50$ with various dimensions. Error bars are $1/3$ of the standard deviation. The abscissa is the \log_2 of the number of simulations, the ordinate is the average of $\log(\|x_n - x^*(f)\|) \times d/n$, supposed to reach $-\log(2) = -0.69315$ asymptotically in n if bound in Eq. 3 is reached.

A prior for which all samples are feasible. We have done our experiments with a prior uniform on all translations of the sphere function. This both simplifies the implementation and allows the comparisons with known mathematical proofs of complexity. On the other hand, a main drawback is that if the fitness function is not a translated sphere, the billiard might be frozen (because there might be no satisfiable solution!). So, an immediate further work consists in generalizing the prior. The approach needs a prior, i.e. a distribution on fitness functions: we can for example consider a fitness function f drawn as follows:

- randomly draw $k \in \mathbb{N}$, $k = i$ with probability $1/2^i$;
- randomly draw x_1, \dots, x_k uniformly and independently in the domain D ;
- then x has fitness $\min_{i \in [1, k]} \|x - x_i\|$.

An immediate consequence of this choice is that the algorithm would never get stuck in a situation in which some points can't be ranked by some f : for any distinct p_1, \dots, p_n , there is a non-zero probability on f that $f(p_1) < f(p_2) < \dots < f(p_n)$. [13] also proposes, with a billiard algorithm, some possible space of functions (using the so-called kernel trick with a Gaussian kernel), so that arbitrary functions can be approximated by their prior.

Interestingly, with such a prior, the algorithm is optimal for a Bayesian prior which is compliant with multimodal optimization.

A population size larger than 2. A second step consists in using comparisons between more than 2 points. This is theoretically easy (Algorithm 1 does not assume that we compare only 2 points and is ready for applications with $\lambda > 2$ points), and important as it provides optimal parallel optimization algorithms as well as this paper proposes optimal sequential optimization algorithms.

8. REFERENCES

- [1] K. Astrom. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.
- [2] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.
- [3] A. Auger. Convergence results for $(1, \lambda)$ -SA-ES using the theory of φ -irreducible markov chains. *Theoretical Computer Science*, 334:35–69, 2005.
- [4] A. Auger, M. Jebalia, and O. Teytaud. Xse: quasi-random mutations for evolution strategies. In *Proceedings of Evolutionary Algorithms*, 12 pages, 2005.
- [5] A. Auger and O. Teytaud. Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica*, Accepted.
- [6] G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy. Progressive strategies for monte-carlo tree search. In P. Wang et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.
- [7] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In P. Ciancarini and H. J. van den Herik, editors, *Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.
- [8] R. Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.
- [9] S. Droste. Not all linear functions are equally difficult for the compact genetic algorithm. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 679–686, 2005.
- [10] S. Gelly, J. B. Hoock, A. Rimmel, O. Teytaud, and Y. Kalemkarian. The parallelization of monte-carlo planning. In *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO 2008)*, pages 198–203, 2008. To appear.
- [11] S. Gelly, S. Ruetten, and O. Teytaud. Comparison-based algorithms: worst-case optimality, optimality w.r.t a bayesian prior, the intra-class-variance minimization in eda, and implementations with billiards. In *PPSN-BTP workshop*, 2006.
- [12] S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press.
- [13] R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1:245–279, 2001.
- [14] J. Jagerskupper and C. Witt. Runtime analysis of a $(\mu+1)$ es for the sphere function. Technical report, 2005.
- [15] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, 1998.

- [16] L. Kocsis and C. Szepesvari. Bandit-based monte-carlo planning. In *ECML'06*, pages 282–293, 2006.
- [17] T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.
- [18] G. Rudolph. Convergence rates of evolutionary algorithms for a class of convex objective functions. *Control and Cybernetics*, 26(3):375–390, 1997.
- [19] T.-P. Runarsson. Ordinal regression in evolutionary computation. In *proceedings of PPSN*, pages 1048–1057, 2006.
- [20] O. Teytaud and H. Fournier. Lower bounds for evolution strategies using VC-dimension. In G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, editors, *PPSN*, volume 5199 of *Lecture Notes in Computer Science*, pages 102–111. Springer, 2008.
- [21] O. Teytaud and S. Gelly. General lower bounds for evolutionary computation. In *proceedings of PPSN*, 2006.
- [22] E. Vazquez, J. Villemonteix, M. Sidorkiewicz, and E. Walter. Global optimization based on noisy evaluations: an empirical study of two statistical approaches. *Journal of Global Optimization*, page 17 pages, 2008.
- [23] J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, page 26 pages, 09 2008.
- [24] Y. Wang and S. Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, pages 175–182, 2007.
- [25] Z. Zhou. Hierarchical surrogate-assisted evolutionary optimization framework. In *In Evolutionary Computation, 2004. CEC2004. Congress on*, pages 1586–1593, 2004.