Upper Confidence Tree for planning restart strategies in Multi-Modal Optimization

Amaury Dubois, Julien Dehos, Fabien Teytaud

Noname manuscript No.
(will be inserted by the editor)

Upper Confidence Tree for planning restart strategies in Multi-Modal Optimization

Amaury Dubois · Julien Dehos · Fabien Teytaud.

the date of receipt and acceptance should be inserted later

Abstract In the context of gradient-free multi-modal optimization, numerous algorithms are based on restarting evolution strategies. Such an algorithm classically performs many local searches, for finding all the global optima of the objective function. The strategy used to select the restarting points (i.e. the initial points of the local searches) is a crucial step of the method. In previous works, a strategy based on reinforcement learning has been proposed: the search space is partitioned and a multi-armed bandit algorithm is used to select an interesting region to sample. This strategy significantly improves the main optimization algorithm but is limited to small dimensional problems.

In this paper, we propose an algorithm tackling this problem. The main contributions are (1) a tree-based scheme for hierarchically partition the search-space and (2) a multi-armed bandit strategy for traversing this tree. Thus, a node in the tree corresponds to a region of the search-space, and its children partition this region according to one dimension. The multi-armed bandit strategy is used to traverse the tree by selecting interesting children recursively. We have experimented our algorithm on difficult multi-modal functions, with small and large dimensions. For small dimensions, we observe performances comparable to previous state-of-the-art algorithms. For large dimensions, we observe better results as well as lower memory consumption.

Amaury Dubois, Julien Dehos, Fabien Teytaud
LISIC, Université du Littoral Côte d'Opale
50, rue F Buisson BP 719
62228 Calais cedex France
E-mail: firstname.lastname@univ-littoral.fr

Amaury Dubois
Weenat Technocampus Alimentation,
2 impasse Therese Bertrand-Fontaine,44320 Nantes

Keywords Multi-modal Optimization · Continuous Optimization · Reinforcement learning · Evolutionary Algorithm

1 Introduction

Multi-Modal Optimization (MMO) is used in numerous engineering applications and scientific fields, including machine learning (Mason et al., 2018; Rapin and Teytaud, 2018), for solving real world problems, which often have multiple solutions. The goal is to find all the optima of a given function, in a minimum number of function evaluations. More formally, given a function $f : [0,1]^D \to \mathbb{R}$, we need to find all the points $\mathbf{x}$ such that $f(\mathbf{x}) = y^*$ where y^* is an optimum value of f. Many algorithms have been proposed for solving MMO problems, using different approaches or different hypothesis about the function to optimize.

In this paper, we consider the general case of multi-dimensional continuous black-box functions, i.e. which can be optimized by derivative-free algorithms only. In this context, Evolution Strategies (ES) have shown their robustness and their efficiency (Beyer, 2001; Rapin and Teytaud, 2018; Rechenberg, 1973). ES belong to the family of evolutionary algorithms, which are biologically inspired. The principle of an Evolution Strategy is to generate a population of individuals (i.e., initial points), then mutate the population (i.e., apply a random variation to the points) and select the best individuals according to the environment pressure (i.e., function evaluation on the points). After several successive mutation/selection iterations, the population converges to a solution. This solution may contain local optima or miss some global optima, but the method can be improved by including a niching technique (Li, 2016).

In real world applications, the function to optimize can be very difficult: large dimensionality, heterogeneous distribution of the optima... These functions are difficult to optimize: MMO algorithms need a huge number of function evaluations and find only a few of the optima. To handle this problem, we propose a new algorithm which partitions the search-space recursively and estimates the interesting regions thanks to a reinforcement learning technique. This algorithm can be seen as an Evolution Strategy which uses an Upper Confidence Tree (Kocsis and Szepesvári, 2006) as a niching technique. Our experiments, using several difficult functions, show that the proposed algorithm learns the interesting regions of the function to optimize. As a result, our algorithm can, with the same number of evaluations, finds more optima than the other methods that we have compared. Our algorithm can also handle functions with a larger dimensionality than our previous algorithm Dubois et al. (2018).

The rest of this paper is organized as follows. In Section 2, we present state-of-the-art MMO algorithms based on ES with niching, or based on a partition of the search-space. In Section 3, we detail the proposed algorithm. In Section 4, we present the experimental results of our algorithm, for several functions, and compare them with the results of the previous algorithms. Finally, we conclude in Section 5.

2 Related work

Evolution Strategies for Multi-Modal Optimization have been widely studied. Most of the proposed algorithms implement a niching technique (Li, 2016; Preuss, 2015), such as crowding (De Jong, 1975; Mahfoud, 1995; Mengshoel et al., 1999), sharing (Goldberg et al., 1987), clearing (Petrowski, 1996; Singh and Deb Dr, 2006) and restarting (Ahrari et al., 2017; Auger and Hansen, 2005; Kadioglu et al., 2017; Teytaud and Teytaud, 2016). Here, the niching technique is very important because it enables the population to evolve and to find niches that contain optima.

In (Schoenauer et al., 2011), Schoenauer et al. propose an ES algorithm with a restarting strategy as a niching technique. The algorithm performs decreasing step-size local searches and restart successive searches from random (or quasi-random) initial points. This algorithm reaches good performances compared to previous niching techniques. However, it does not take into account the landscape of the function, since the restarting points are randomly sampled in the whole search-space. Thus, for difficult functions (large dimension, heterogeneous distribution of the optima...), the algorithm may need a great number of function evaluations to find all the optima.

For difficult functions, the restart strategy, i.e. the choice of the initial point for a new local search, can greatly impact the global performance of the algorithm. In (Dubois et al., 2018), we proposed an improved algorithm to learn the interesting regions for sampling initial points. In that work, we partition the search-space in regular regions and model the choice of a region (to sample restarting points) as a Multi-Armed Bandit problem. Thus, the algorithm automatically learns the interesting regions of the function and outperforms classic restarting-based algorithms. However, the partition scheme is constant during the execution of the algorithm so the algorithm can only roughly learn the landscape of the function. Moreover, the number of bandit arms (i.e., regions of the search-space) grows exponentially with the dimension, which limits the algorithm to small dimensions only. The contributions proposed in the next sections are a continuation of this work.

In (Munos, 2011), Munos proposes a deterministic tree-based algorithm for mono-modal optimization problems. This algorithm searches the global optimum of the function by incrementally building a hierarchical partitioning of the search-space. At each iteration, it selects the current best leaf node (according to the fitness of the function at the mid-point of the corresponding region), then it expands the selected node (i.e., partitions the corresponding region). The algorithm can find the global optimum with no knowledge of the smoothness of the function but it has poor performances on complex functions or when the required precision is high (Preux et al., 2014). In (Valko et al., 2013), Valko et al. propose a modification of the algorithm, where the nodes are selected using a stochastic method that constructs upper confidence bounds over the partitions of the search-space. However, the algorithm does not implement a local search in the sampled regions and does not handle multi-modal optimization problems.

Finally, other algorithms, such as Ant Colony Optimization and Particle Search Optimization, have also been used for Multi-Modal Optimization (Deng et al., 2019; Deng et al., 2019; Deng et al., 2017; Zhao et al., 2019, 2020).

3 Model and algorithm

In this section, we first recall the Random restarts with Decreasing Step-size algorithm (RDS). Then, we present the reinforcement learning model considered in this paper and the Upper Confidence Tree method (UCT). Finally, we detail the proposed algorithm (UCT-RDS)

which uses the UCT method to improve the RDS algorithm, for Multi-Modal Optimization problems.

3.1 Random restarts with Decreasing Step-size

Random restarts with Decreasing Step-size (Schoenauer et al., 2011) is an Evolution-Strategy-based Multi-Modal Optimization algorithm which uses the restarting technique. It consists in a single-optimum local search, controlled by a restart strategy.

Algorithm 1: SearchDS

{Search an optimum using a Decreasing Step-size }
Input:
f: function to optimize
σ_0: initial step-size
ϵ_σ: threshold value of the step-size
y^*: maximum fitness of the function
ϵ_y: threshold value of the fitness
$\mathbf{x}$: initial position for the search
ϵ_x: threshold value of the position
$\hat{\mathbf{X}}$: set of previously found optima
Output:
$\hat{\mathbf{X}}$: updated set of optima

begin
  $y \leftarrow f(\mathbf{x})$
  $\sigma \leftarrow \sigma_0$
  repeat
    {mutation}
    $\mathbf{x}' \leftarrow \mathcal{N}(\mathbf{x}, \sigma)$
    $y' \leftarrow f(\mathbf{x}')$
    {selection with 1/5th adaptation}
    if $y' > y$ **then**
      $\mathbf{x} \leftarrow \mathbf{x}'$
      $\sigma \leftarrow 2\sigma$
    else
      $\sigma \leftarrow 2^{-1/4}\sigma$
    {discard search if optimum already known}
    if $\exists \hat{\mathbf{x}} \in \hat{\mathbf{X}}, \|\mathbf{x} - \hat{\mathbf{x}}\| < \epsilon_x$ **then**
      break
    {store found optimum}
    if $\|y - y^*\| < \epsilon_y$ **then**
      $\hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}} \cup \{\mathbf{x}\}$
      break
  until $\sigma < \epsilon_\sigma$ *or max nb of function evaluations reached*

The local search is a simple (1+1)-ES with the 1/5 adaptation rule (see Algorithm 1). This algorithm evolves candidate solutions iteratively: it generates a new point by mutating the current point (using a normal distribution with standard deviation σ) and selects the best of these two points. If the new mutated point is better, the step-size σ is increased because this means that the current point is far from the solution, so the search must be extended to a larger neighborhood. Otherwise, σ is decreased because this means that the current point is close to the solution, so the search must focus on a smaller neighborhood. The search is terminated if the current point is converging to an optimum already found in a previous search or when the current point has converged to a new optimum (and, in this case, the new optimum is stored). Here, we consider that the value of the global optima is known but it is very easy to modify the algorithm if this value is unknown.

Algorithm 2: RDS

{Random restarts with Decreasing Step-size}
Input: $f, \sigma_0, \epsilon_\sigma, y^*, \epsilon_y, \epsilon_x$
Output: $\hat{\mathbf{X}}$

begin
  $\hat{\mathbf{X}} \leftarrow \varnothing$
  repeat
    $\mathbf{x} \leftarrow$ sample search-space
    $\hat{\mathbf{X}} \leftarrow \text{SearchDS}(f, \sigma_0, \epsilon_\sigma, y^*, \epsilon_y, \mathbf{x}, \epsilon_x, \hat{\mathbf{X}})$
  until *all optima found or max nb of function evaluations reached*

Using this local search algorithm, a restart strategy is simple to implement (see Algorithm 2): at each iteration, an initial point is chosen in the search-space and is used as the starting point of a local search. When a local search is terminated, a new local search is started, if all the optima are not found and if the maximum number of function evaluations is not reached.

In (Schoenauer et al., 2011), Schoenauer et al. compare two algorithms: Random restarts with Decreasing Step-size (RDS) and Quasi-Random restarts with Decreasing Step-size (QRDS). They show that sampling restarting points according to a quasi-random sequence rather than a purely random sequence is more efficient.

In this paper, we propose an algorithm that learns the interesting regions to sample. This algorithm is based on reinforcement learning as explained below.

3.2 Reinforcement learning

Reinforcement learning is a field of machine learning that considers how agents can take actions in an environment. The main idea is to reward the actions of the agent. This can be modeled as a Markov Decision Process $M = (\mathcal{S}, \mathcal{A}, P, R)$, where:

- $\mathcal{S}$ is the set of states,
- $\mathcal{A}$ is the set of actions,
- $P_a(s, s') = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability of transition from state s to s' under action a (at discrete time step t),

– $R_a(s, s')$ is the reward after transition from s to s' with action a.

The action selection of the agent is modeled by the *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. This policy gives the probability of making action a when the agent is in state s, i.e. $\pi(s, a) = \mathbb{P}(a_t = a | s_t = s)$.

The *return* R_t is the cumulative reward, for the current episode, from time t until a final state is reached, at time T (Gelly and Silver, 2007).

$$R_t = \sum_{k=t+1}^{T} r_k \tag{1}$$

Value-based reinforcement learning methods attempt to find the best policy, i.e. with maximum expected return, by computing an intermediate *action value function*.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] \tag{2}$$

In other words, this function estimates the return when the agent takes action a in state s and uses policy π for taking all subsequent actions.

The action value function is generally estimated using an iterative algorithm: an estimation of the function can be used to improve the policy and therefore compute a new estimation of the function. At each step of this process, the algorithm has to choose an action to consider, i.e. the well-known tradeoff between exploitation (selecting the maximum value action) and exploration (selecting a random action). This tradeoff is a classic Multi-Armed Bandit problem (Auer et al., 2002a).

3.3 Upper Confidence Tree

The Upper Confidence Tree (UCT) algorithm is a value-based reinforcement learning algorithm (Silver, 2009) (Kocsis and Szepesvári, 2006). It is the most famous implementation of Monte-Carlo Tree Search (MCTS).

The UCT algorithm estimates an action value function $Q_{UCT}(s, a)$ by building a search tree where the root node corresponds to the start state and the subtrees correspond to the subsequent visited states. In other words, the tree $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A}$ is a subset of all the (state, action) pairs. The value of each node is estimated by Monte Carlo simulation.

The policy used in the UCT algorithm is based on the UCB algorithm (Auer et al., 2002a), i.e. a Multi-Armed Bandit algorithm, designed to solve the exploitation versus exploration problem. For a current state s, if all actions are in the tree ($\forall a \in \mathcal{A}(s), (s, a) \in \mathcal{T}$) then the tree-policy is applied and the action that maximises the Upper Confidence Bound on $Q_{UCT}(a, s)$ is selected:

$$\pi_{UCT}(s) = \arg\max_{a} \; Q_{UCT}(s, a) + k_{UCT} \sqrt{\frac{\log N(s)}{N(s, a)}} \tag{3}$$

where $N(s, a)$ is the number of times action a has been selected from state s, $N(s)$ is total number of selections of s ($N(s) = \sum_{a \in \mathcal{A}(s)} N(s, a)$) and k_{UCT} is a parameter of the algorithm. If any action from the current state is not in the tree ($\bar{\mathcal{A}}(s) = \{a | (s, a) \notin \mathcal{T}\}$ is not empty) then a random action is selected in $\bar{\mathcal{A}}(s)$ and a new node is added in the tree.

Finally, at the end of the episode, each visited node is updated using the return of the episode:

$$\begin{aligned} N(s_t, a_t) &\leftarrow N(s_t, a_t) + 1 \\ Q_{UCT}(s_t, a_t) &\leftarrow Q_{UCT}(s_t, a_t) + \frac{R_t - Q_{UCT}(s_t, a_t)}{N(s_t, a_t)} \end{aligned} \tag{4}$$

New nodes are simply initialized with the return R_t.

Thus, after many episodes, we obtain an unbalanced tree where the more interesting nodes have been more frequently visited (therefore more precisely estimated).

3.4 Planning random restarts with UCT

We propose a derivative-free Multi-Modal Optimization algorithm called UCT-RDS (Upper Confidence Tree for Random restarts with Decreasing Step-size). This algorithm is an improvement of the RDS algorithm that learns which region of the search-space is interesting to sample for restarting a local search. This is implemented by a UCT reinforcement learning method which computes a hierarchical partitioning of the search-space.

As the RDS algorithm and the QRDS algorithm, UCT-RDS iteratively computes local searches (see Algorithm 1) until all the optima have been found or until the maximum number of function evaluations has been reached. However, the restarting points, for these local searches, are not sampled in the whole search-space but into subregions.

The main objective of the algorithm is to select interesting regions while still exploring the whole search space. Moreover, it should be usable for large dimensional functions, i.e. the number of regions should not grows exponentially with the dimension. To this end, we use a hierarchical partitioning similar to (Munos, 2011): the algorithm iteratively selects a region and splits it in K subregions according to one dimension (see Fig. 1a). These successive splits form a hierarchical partitioning of the search-space that can be modeled with a tree (see

Fig. 1b). Thus, if we traverse the tree to a leaf, we get a region that we can sample, for restarting a local search. To select an interesting region while exploring the whole search-space, the UCT-RDS algorithm implements this tree as an Upper Confidence Tree.

The UCT-RDS algorithm is detailed in Algorithm 3. It is a classic UCT algorithm where the states correspond to regions of the search-space and the default-policy is a local search. Initially, a root node, corresponding to the whole search-space, is created then the tree is built iteratively by repeating the four following steps. In the selection step, the tree is traversed to a leaf node by applying the tree-policy. In the expansion step, a new node is created and the selected region is split. In the simulation step, the local search is performed. Finally, in the backpropagation step, all the visited nodes are updated according to the result of the simulation step.

For example in Fig 1, the initialization of the algorithm partitions the search-space horizontally, giving the nodes 1, 2 and 3. Then, the first iteration of the loop traverses the tree, selects the node 1 and splits its search-space vertically, giving the nodes 11, 12 and 13. Finally, the second iteration of the loop traverses the tree, selects the node 12 and splits its search-space horizontally, giving the nodes 121, 122 and 123. During this process, a local search is performed when a new node is created.

Algorithm 3: UCT-RDS

{Upper Confidence Tree for Random restarts with Decreasing Step-size}
Input: $f, \sigma_0, \epsilon_\sigma, y^*, \epsilon_y, \epsilon_x$
Output: $\hat{\mathbf{X}}$

begin
  $\hat{\mathbf{X}} \leftarrow \varnothing$
  { initialization }
  $s_0 \leftarrow$ create root node (corresponding to the whole search-space)
  $\bar{\mathcal{A}}(s_0) \leftarrow$ split the region of s_0 according to one dimension
  { UCT episodes }
  repeat
    { selection }
    $s_t \leftarrow s_0$
    while $\bar{\mathcal{A}}(s_t) = \varnothing$ **do**
      $s_t \leftarrow \arg\max_a \; Q_{UCT}(s_t, a) + k_{UCT}\sqrt{\frac{\log N(s_t)}{N(s_t,a)}}$
      $a_t \leftarrow$ the selected a
    { expansion }
    $s_T \leftarrow$ create a child node of s_t from any $a_t \in \bar{\mathcal{A}}(s_t)$
    $\bar{\mathcal{A}}(s_T) \leftarrow$ split the region of s_T according to one dimension
    { simulation }
    $\mathbf{x} \leftarrow$ sample the region of s_T
    $\hat{\mathbf{X}} \leftarrow \text{SearchDS}(f, \sigma_0, \epsilon_\sigma, y^*, \epsilon_y, \mathbf{x}, \epsilon_x, \hat{\mathbf{X}})$
    { backpropagation }
    $r_T \leftarrow 1$ if a new optima has been found, 0 otherwise
    repeat
      $N_{s_t,a_t} \leftarrow N_{s_t,a_t} + 1$
      $Q_{UCT}(s_t, a_t) \leftarrow Q_{UCT}(s_t, a_t) + \frac{R_t - Q_{UCT}(s_t,a_t)}{N(s_t,a_t)}$
      $s_t \leftarrow$ parent node of s_t
    until $s_t = s_0$
  until *all optima found or max nb of function evaluations reached*

4 Experiments

In order to evaluate the efficiency of our improvement, we have experimented the proposed algorithm UCT-RDS on three multi-modal functions and compared with the QRDS algorithm. In fact, we have also considered various simple synthetic functions such as sphere, cigar and sinus. In this paper, we present our results on more realistic functions (f_{Hump}, f_{HumpSin} and f_{Icop}) since they represent more faithfully real-world problems, where the optima are not regularly distributed in the search-space. Moreover, these functions are often used in the litterature (Dubois et al., 2018; Lacroix et al., 2017; Schoenauer et al., 2011; Singh and Deb Dr, 2006), which enables us to compare the proposed algorithm to existing ones.

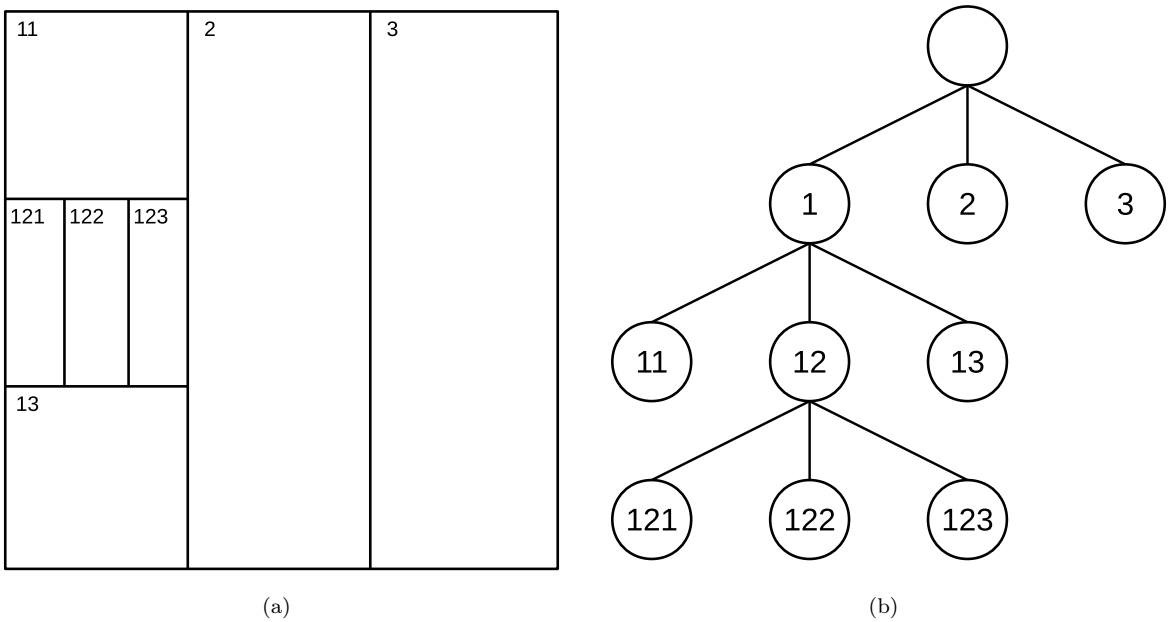
4.1 Multi-modal functions

The first function we used in our experiments, f_{Hump}, is a simplified version of the function presented in (Singh and Deb Dr, 2006), where all the random peaks have the same shape (see Eq. 5 and Fig. 2a).

$$f_{\text{Hump}}(\mathbf{x}) = h \max\left[\, 0,\; 1 - \left(\frac{\min_{q \in Q} \|\mathbf{x} - \mathbf{x_q}\|}{r}\right)^{\alpha} \,\right] \tag{5}$$

Here, α defines the (inverse) sharpness of the peaks, r the radius of the peaks and Q the number of peaks (and therefore the number of optima). The points $\mathbf{x_q}$ define the centers of the peaks and are randomly drawn in $[0,1]^D$. Following (Dubois et al., 2018; Schoenauer et al., 2011; Singh and Deb Dr, 2006), we use $h = 1$, $r = 1.45$ and $\alpha = 1$ in all our experiments. A strong advantage of this function is that, due to its randomness, it cannot be overfitted.

The second function, f_{HumpSin}, has been proposed in (Dubois et al., 2018). It is a variation of f_{Hump} where



(a) (b)

Fig. 1: Illustration of the UCT-RDS partitioning, for a 2D search-space and a local split into $K = 3$ regions. The algorithm partitions a region according to one dimension, hierarchically (a) and builds a corresponding tree (b).

the peaks have a more complicated shape. The randomness advantage of f_{Hump} is kept, but instead of having simple single-optimum peaks, f_{HumpSin} have D-dimensional sine peaks (see Eq. 6, Eq. 7 and Fig. 2b).

$$f_{\text{HumpSin}}(\mathbf{x}) = \begin{cases} f_{\text{Sin}}(\frac{\mathbf{x}-\mathbf{x_z}+r}{2r}) & \text{if } \|\mathbf{x}-\mathbf{x_z}\|_\infty < r \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$f_{\text{Sin}}(\mathbf{x}) = \frac{1}{D}\sum_{d=1}^{D} \sin^{2s}(p\pi x_d) \quad (7)$$

In the definition of f_{HumpSin}, z is the number of local peak zones, and r the radius of the zones. The points $\mathbf{x_z}$ define the centers of the zones and are randomly drawn. In the local D-dimensional sine function f_{Sin}, p is the number of peaks (i.e. optima) per zone. The total number of optima of f_{HumpSin} is then $z \times p^D$. This function is then far more difficult than f_{Hump}, and is a real challenge for MMO algorithms. In our experiments, we use $p = 2$.

The last function, f_{Icop}, comes from (Lacroix et al., 2017). It defines a class of optimization problems with tunable landscape features (see Eq. 8 and Fig. 2c).

$$f_{\text{Icop}}(\mathbf{x}) = \begin{cases} u_i & \text{if } \exists i, ||\mathbf{x}-\mathbf{x}_i|| \approx 0 \\ \frac{\sum_{i=1}^{\Omega} \frac{u_i}{||\mathbf{x}-\mathbf{x}_i||^p}}{\sum_{i=1}^{\Omega} \frac{1}{||\mathbf{x}-\mathbf{x}_i||^p}} & \text{otherwise} \end{cases} \quad (8)$$

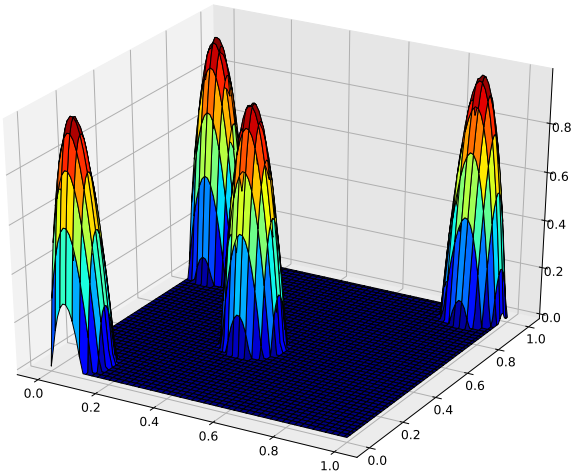
To create a f_{Icop} function, we generate Ω global optima $\mathbf{x_i}$, with respective fitnesses u_i. If we want a multimodal function, we can set half of the $\mathbf{x_i}$ as global optima (i.e. $u_i = 1$) and the other half as local optima (i.e. $0 \leq u_i \leq u_l < 1$). The purpose of Icop functions is to have a generic and tunable class of optimization problems:

- Ω: number of optima (local or global),
- u_i: fitness of the i-th global optimum,
- u_l: maximum fitness allowed for local optima,
- p: influence of nearby seed solutions on the interpolation.

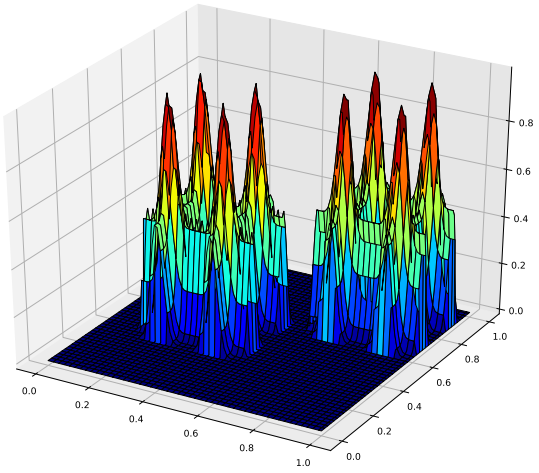
In our experiments, we use $u_i = 1$, $p = 1$, and we have tested different values for Ω and u_l.

4.2 Results

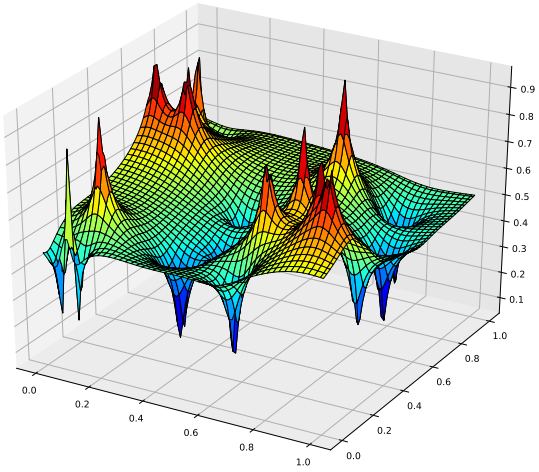
We have compared the proposed algorithm, UCT-RDS, with QRDS (Schoenauer et al., 2011), a state-of-the-art multi-modal optimization algorithm, using the functions defined in the previous section. Each experiment has been performed 100 times. The number of evaluations allocated to each function is 10^6. For the local searches, the threshold value of the fitness is $\epsilon_y = 0.00001$ and the threshold value of the position is $\epsilon_x = 0.001$. To adjust the parameters of the proposed algorithm, we have carried out an empirical study and we have selected the parameters giving the best performance. The results reported below correspond to the



(a) f_{Hump} with $\alpha = 1.0$, $r = 0.1$, $Q = 4$ (see Eq. 5).



(b) f_{HumpSin} with $s = 4$, $p = 2$, $z = 2$ and $r = 0.2$ (see Eq. 6).



(c) f_{Icop} with $u_i = 1.0$, $\Omega = 20$, $p = 1.0$ (see Eq. 8).

Fig. 2: Multi-modal functions used in our experiments (depicted in 2D).

average and the standard error ϵ, defined as $\epsilon = \sigma/\sqrt{n}$, where σ is the standard deviation and n the number of runs (in our experiments, $n = 100$).

The first set of experiments consists in finding the maximum number of optima within a given number of function evaluations (see Table 1). First, on the f_{Icop} function, the results with our UCT-RDS algorithm are at least as good as the results with the QRDS algorithm. It is important to note that f_{Icop} is a test of robustness for our algorithm: by definition, there is no predefined area which should be more important than another one, as the global optima are randomly drawn. Our algorithm is then quite robust since it reaches the performances of QRDS. UCT-RDS is even better on some sets of parameters, because if a group of optima is drawn in a same area, then UCT can learn that this area is important.

Similar results are seen on the f_{Hump} function. We have tested a really hard set of parameters (dimension 35 and 50 optima) and found better results for UCT-RDS which has found 7 more optima than QRDS, in average.

On the f_{HumpSin} function, the results are very significant. We recall that this function is very difficult and that previous algorithms are very limited on it (Dubois et al., 2018). With UCT-RDS, we are able to find 314 of the 2048 optima, in dimension 5, whereas QRDS is only able to find 12 optima, in average. In dimension 8, with somehow easier parameters, UCT-RDS is able to find twice as many optima as QRDS (67.5 vs 32.9, in average).

Finally, in Fig. 3, we study the influence of the parameters of UCT-RDS, on the f_{HumpSin} function in dimension 8. This algorithm has two parameters: the number of subregions K a region is split and the *exploration vs exploitation* trade-off k_{UCT}. First, we can see that the number of subregions is important. In our experiments, we have noticed that K has to slowly grow with the dimension. Second, the k_{UCT} parameter is very important. A large parameter tends to explore more, while a small one tends to exploit more. It is known that the optimal strategy is somewhere between extreme exploration and extreme exploitation, which is also verified in our experiments.

5 Conclusion

In this paper, we proposed a derivative-free Multi-Modal Optimization algorithm. This algorithm implements an Upper Confidence Tree that hierarchically partitions the search-space and computes a local Decreasing Stepsize search. We model the successive selections of the subregion to search as a Multi-Armed Bandit problem. This enables the algorithm to learn which subregions of the search-space are interesting, in order to spend more function evaluations on it (local search). We experimented our algorithm on three strongly multi-modal functions and verify the robustness of our method regardless the dimension and the function landscape. Compared to previous approaches, we observed significantly better results on all the tested functions for both small and large dimensions.

This method requires the tuning of two parameters, k_{UCT} (exploration vs exploitation trade-off) and K (number of subregions in which a region must be split). As future work, we will focus on tuning these two parameters automatically. The tuning of k_{UCT} should be done thanks to the UCB-Tuned strategy (Auer et al., 2002b). Automatically choosing a good value of K is another challenge: using the variance of the subregion may be an appropriate solution. Adding improvements, such as using the local smoothness properties of functions (Bubeck et al., 2011), may also improve the performances of our algorithm.

Acknowledgements Experiments presented in this paper were carried out using the CALCULCO computing platform, supported by SCOSI/ULCO (Service COmmun du Système d'Information de l'Université du Littoral Côte d'Opale).

Compliance with ethical standards

Conflict of interest
Amaury Dubois has received research grants from *Weenat*. Julien Dehos and Fabien Teytaud declare that they have no conflict of interest.

Ethical approval
This article does not contain any studies with human participants or animals performed by any of the authors.

References

Ahrari, A., Deb, K., and Preuss, M. (2017). Multimodal optimization by covariance matrix self-adaptation evolution strategy with repelling subpopulations. *Evolutionary computation*, 25(3):439–471.

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

Table 1: Average number of optima found by each algorithm after 10^6 evaluations, over 100 runs (higher is better). We have experimented different parameters, in particular for f_{Icop}, in order to evaluate the robustness of UCT-RDS.

Function	***D***	**QRDS**	**UCT-RDS**	**#Optima**
f_{HumpSin} $[z = 4, r = 0.1]$	5	12.31 ± 0.25	**314.64** ± 6.2 $(K = 3,\ k_{UCT} = 0.1)$	2048
f_{Icop} $[\Omega = 50, u_l = 0.9]$	5	11.64 ± 0.3	**12.6** ± 0.25 $(K = 2,\ k_{UCT} = 6.4)$	25
f_{Icop} $[\Omega = 80, u_l = 0.9]$	5	12.7 ± 0.1	**13.6** ± 0.1 $(K = 2,\ k_{UCT} = 1.0)$	40
f_{HumpSin} $[z = 2, r = 0.22]$	8	32.9 ± 0.3	**67.5** ± 0.8 $(K = 2,\ k_{UCT} = 0.8)$	512
f_{Icop} $[\Omega = 100, u_l = 0.25]$	10	34.6 ± 3.4	**39.2** ± 3.9 $(K = 7,\ k_{UCT} = 1.4)$	50
f_{Icop} $[\Omega = 96, u_l = 0.25]$	10	34.6 ± 0.4	**35.9** ± 0.8 $(K = 2,\ k_{UCT} = 6.4)$	48
f_{Icop} $[\Omega = 128, u_l = 0.25]$	10	40.44 ± 0.8	40.4 ± 0.8 $(K = 3,\ k_{UCT} = 1.4)$	64
f_{Icop} $[\Omega = 144, u_l = 0.25]$	10	40.48 ± 0.9	40.45 ± 0.84 $(K = 2,\ k_{UCT} = 1.4)$	72
f_{Icop} $[\Omega = 24, u_l = 0.25]$	20	11.09 ± 0.1	11.24 ± 0.1 $(K = 2,\ k_{UCT} = 12.4)$	12
f_{Icop} $[\Omega = 60, u_l = 0.25]$	20	13.87 ± 0.4	13.94 ± 0.41 $(K = 9,\ k_{UCT} = 0.4)$	30
f_{Hump}	35	23.21 ± 0.3	**30.21** ± 0.33 $(K = 13,\ k_{UCT} = 0.1)$	50

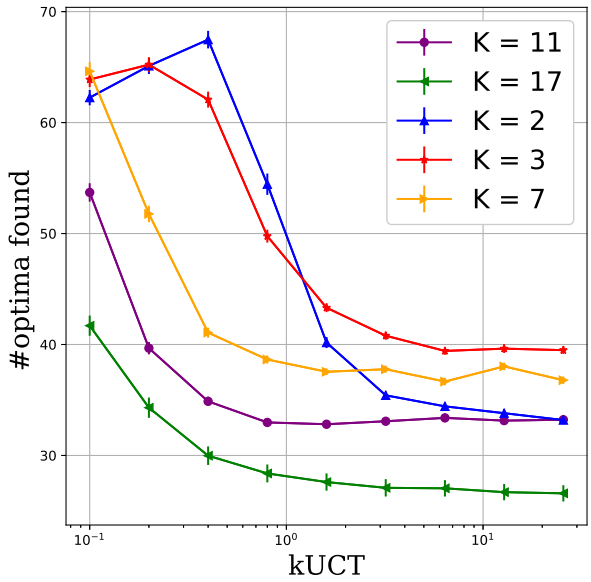


Fig. 3: Number of optima found in front of k_{UCT} (in log scale), when splitting a region in K subregions, on f_{HumpSin} and in dimension 8.

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

Auger, A. and Hansen, N. (2005). A restart cma evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1769–1776.

Beyer, H.-G. (2001). *The Theory of Evolution Strategies*. Springer Science & Business Media.

Bubeck, S., Munos, R., Stoltz, G., and Szepesvári, C. (2011). X-armed bandits. *Journal of Machine Learning Research*, 12(May):1655–1695.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.

Deng, W., Xu, J., Song, Y., and Zhao, H. (2019). An effective improved co-evolution ant colony optimization algorithm with multi-strategies and its application. *Int. J. Bio-Inspired Comput.*

Deng, W., Xu, J., and Zhao, H. (2019). An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. *IEEE Access*, 7:20281–20292.

Deng, W., Zhao, H., Yang, X., Xiong, J., Sun, M., and Li, B. (2017). Study on an improved adaptive pso algorithm for solving multi-objective gate assignment. *Applied Soft Computing*, 59:288 – 302.

Dubois, A., Dehos, J., and Teytaud, F. (2018). Improving Multi-Modal Optimization Restart Strategy Through Multi-Armed Bandit. In *IEEE ICMLA 2018: 17th IEEE International Conference On Machine Learning And Applications*.

Gelly, S. and Silver, D. (2007). Combining Online and Offline Knowledge in UCT. In *International Conference of Machine Learning*.

Goldberg, D. E., Richardson, J., et al. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49.

Kadioglu, S., Sellmann, M., and Wagner, M. (2017). Learning a reactive restart strategy to improve stochastic search. In *International Conference on Learning and Intelligent Optimization*, pages 109–123.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293.

Lacroix, B., Christie, L. A., and Mcall, J. A. (2017). Interpolated continuous optimisation problems with tunable landscape features. In *Proc. of GECCO '17 Companion*.

Li, X. (2016). *Multimodal Optimization using Niching Methods*, pages 1–8. American Cancer Society.

Mahfoud, S. W. (1995). Niching methods for genetic algorithms. *Urbana*, 51:62–94.

Mason, K., Duggan, J., and Howley, E. (2018). Maze navigation using neural networks evolved with novelty search and differential evolution. In *Adaptive and Learning Agents Workshop (at ICML-AAMAS 2018)*.

Mengshoel, O. J., Goldberg, D. E., et al. (1999). Probabilistic crowding: Deterministic crowding with probabilistic replacement. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 409–416. Morgan Kauffman.

Munos, R. (2011). Optimistic optimization of deterministic functions without the knowledge of its smoothness. *Advances in Neural Information Processing Systems*, pages 783–791.

Petrowski, A. (1996). A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 798–803.

Preuss, M. (2015). *Multimodal Optimization by Means of Evolutionary Algorithms*. Springer Publishing Company, Incorporated, 1st edition.

Preux, P., Munos, R., and Valko, M. (2014). Bandits attack function optimization. In *IEEE Congress on Evolutionary Computation*.

Rapin, J. and Teytaud, O. (2018). Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Number 15 in Problemata. Frommann-Holzboog.

Schoenauer, M., Teytaud, F., and Teytaud, O. (2011). A Rigorous Runtime Analysis for Quasi-Random Restarts and Decreasing Stepsize. In *Artificial Evolution*, Angers, France.

Silver, D. (2009). *Reinforcement Learning and Simulation-Based Search*. PhD thesis, University of Alberta.

Singh, G. and Deb Dr, K. (2006). Comparison of multimodal optimization algorithms based on evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1305–1312.

Teytaud, F. and Teytaud, O. (2016). Qr mutations improve many evolution strategies: A lot on highly multimodal problems. In *Proc. of the 2016 GECCO conference*, pages 35–36.

Valko, M., Carpentier, A., and Munos, R. (2013). Stochastic simultaneous optimistic optimization. In

Proc. of the 30th International Conference on Machine Learning, volume 28.

Zhao, H., Liu, H., Xu, J., and Deng, W. (2019). Performance prediction using high-order differential mathematical morphology gradient spectrum entropy and extreme learning machine. *IEEE Transactions on Instrumentation and Measurement*.

Zhao, H., Zheng, J., Deng, W., and Song, Y. (2020). Semi-supervised broad learning system based on manifold regularization and broad network. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(3):983–994.