Improving Multi-Modal Optimization Restart Strategy Through Multi-Armed Bandit

Amaury Dubois, Julien Dehos, Fabien Teytaud

Improving Multi-Modal Optimization Restart Strategy Through Multi-Armed Bandit

Amaury Dubois
LISIC
Université du Littoral Côte d'Opale
dubois.amaury62@gmail.com

Julien Dehos
LISIC
Université du Littoral Côte d'Opale
julien.dehos@univ-littoral.fr

Fabien Teytaud
LISIC
Université du Littoral Côte d'Opale
fabien.teytaud@univ-littoral.fr

***Abstract*—Multi-Modal Optimization problems are widespread and can be solved using numerous methods, such as niching, sharing or clearing. In this paper, we are interested in algorithms based on restart strategies, where the searching point is restarted at another initial position when an optimum is found. Previous works show that the choice of these initial positions greatly impacts the performance of the algorithm but is not easy to make. In this paper, we propose a new restart strategy, based on reinforcement learning. Our algorithm subdivides the search space and uses a Multi-Armed Bandit technique to choose the successive restart positions. We experiment this algorithm on various functions and on a modified Hump function with more complex local areas. Our results show significant improvements over previous algorithms, such as the Quasi-Random restart with Decreasing Step-size algorithm.**

***Index Terms*—Multi-modal optimization, reinforcement learning, multi-armed bandit, evolution strategy.**

I. Introduction

Multi-Modal Optimization (MMO) is a very important tool for numerous applications in engineering or in machine learning. In this work, we are interested in finding all the global optima (maxima) of a given but unknown function. More precisely, we have a multi-dimensional continuous black-box function $f : [0,1]^D \to \mathbb{R}$ and we want to find all the points $\mathbf{x} \in [0,1]^D$ such that $f(\mathbf{x}) = y^*$ where y^* is the maximum value (fitness) of f. This kind of optimization is related to classic mono-modal optimization when local optima are very close to the global optimum: in this case, it is necessary to find all the local maxima in order to determine the global one. To analyze the performance of a MMO algorithm, we generally consider the number of evaluations of f: a good algorithm finds all the optima with few evaluations.

Numerous MMO algorithms have been proposed but many of them use the derivative of the function (gradient-based methods), which is not applicable in the context of black-box functions. In this context, gradient-free methods are generally based on Evolution Strategies (ES) [1], [2]. ES algorithms consist in choosing an initial point (also called individual) randomly then mutating this point to a better one. In the context of MMO, ES algorithms are combined either with a niching technique or a restart strategy in order to find all the optima.

Amongst the various niching techniques, the sharing technique [3] states that all individuals of a same niche (i.e. a group of individuals which are close to each other) share a same fitness value. This tends to force the algorithm to explore new areas of the search space. The crowding technique [4] (including deterministic crowding [5] and probabilistic crowding [6]) consists in replacing individuals of the current population by similar candidate individuals of a new population. The clustering technique [7] consists in analyzing and grouping similar individuals. Finally, the clearing technique [8] and more recently, the modified clearing technique [9], are variants of the sharing technique where the best individuals are preserved (dominant individuals) while the fitness of the others is cleared (dominated individuals).

Another classic MMO technique is the restarting technique [10]–[13]. The algorithm proposed in [14] uses a quasi-random restart strategy with a decreasing step-size search. This algorithm has good performances when the optima are quite equally distributed over the search space or when the function is locally simple. In this paper, we propose a more flexible algorithm, where the restart strategy automatically learns which areas of the search space are interesting.

The rest of this paper is organized as follows. Section II and Section III, recall the quasi-random restart strategy and the multi-armed bandit learning method. Then, we detail the proposed method in Section IV and present experimental results in Section V. Finally, we conclude in Section VI.

II. QRDS optimization

Quasi-random Restart with Decreasing Step-size (QRDS) [14] is an Evolution-Strategy-based Multi-Modal Optimization algorithm which uses the restarting technique. It consists in a single-optimum local search (see Algorithm 1), controlled by a restart strategy (see Algorithm 2).

The restart strategy (Algorithm 2) is very simple: at each iteration, an initial point is chosen randomly in the search space, then used as the starting point of a local search. Once this local search is finished (and the set of found optima is updated), a new iteration is started unless all optima are found. As stated in [14], it is generally more efficient to sample the initial points using a quasi-random method (SampleQR) than using a purely-random method. In this paper, we note the quasi-random method as QRDS and the purely-random method as RDS.

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
y: value of the newly found optimum

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 $1/5$th 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 $\sigma < \epsilon_\sigma$

Algorithm 2: QRDS

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

begin
  $\hat{\mathbf{X}} \leftarrow \varnothing$
  while *all optima not found* **do**
    $\mathbf{x} \leftarrow$ SampleQR()
    $\hat{\mathbf{X}}, y \leftarrow$ SearchDS$(f, \sigma_0, \epsilon_\sigma, y^*, \epsilon_y, \mathbf{x}, \epsilon_x, \hat{\mathbf{X}})$

For the local search, a simple (1+1)-ES with the 1/5th adaptation rule can be used (Algorithm 1), as stated in [14]. This algorithm iteratively mutates the current point (using a normal distribution with standard deviation σ) and selects the best of these two points. If the current point is better, the search step-size σ is decreased (the current point is interesting so the search should continue in a more local area). If the mutated point is better, σ is increased (the current point is not interesting so the search should continue in another area). The search is terminated when the current point has converged to a new optimum (in this case, the optimum is added to the set of found optima), to an already known optimum or to a local optimum.

The QRDS algorithm is known to give good results when the optima are quite regularly distributed or when the function is locally simple. In this paper, we are interested in optimizing more complex functions (with basins or with complex local areas) by modelling the restart strategy as a multi-armed bandit problem.

III. Multi-Armed Bandit

Multi-Armed Bandit (MAB) [15] is a classic reinforcement learning problem, where a player should successively select one bandit arm, with unknown reward, in order to maximize the total cumulated reward.

More precisely, N bandit arms with unknown reward probabilities p_n are available. At each time step k, the player selects an arm and receives a reward $r_k = 1$ with probability p_n or $r_k = 0$ otherwise. The goal is to maximize the total reward gathered over all the K time steps. In other words, the player has to find the best policy in order to minimize the loss compared to the best arm. This regret is defined as:

$$Kp^* - \sum_{k=1}^{K} r_k, \tag{1}$$

where $p^* = \max\{p_1, \ldots, p_N\}$ is the maximal reward probability among the bandit arms.

The Upper Confidence Bound (UCB) algorithm [16] is a state of the art method to deal with the MAB problem, since it provides an optimal asymptotic bound on the regret in $\mathcal{O}(\ln(K))$. At each time step k, the UCB algorithm selects the arm j which maximizes:

$$\hat{p}_{j,k} + \sqrt{\frac{2\ln\left(\sum_{n=1}^{N} A_{n,k}\right)}{A_{j,k}}}, \tag{2}$$

where $\hat{p}_{j,k}$ is the average reward for the arm j and $A_{j,k}$ is the number of times the arm j has already been selected.

This formula deals with the well-known trade-off between *exploitation* and *exploration*. Exploitation (the first part of the formula) tends to select the arm with the optimal average reward. Exploration (the second part of the formula) tends to select the arm which has been selected the most rarely.

IV. Proposed method

Restart-based MMO algorithms, such as QRDS, select new starting points in the whole search space, using a (purely-random or quasi-random) uniform sampling method. This approach suits to locally simple functions and to functions where the optima are regularly distributed. However, in many real-world applications, the function to optimize is more complex (locally complex, basins…). Therefore, some areas of the search space are more important to consider than others.

In this paper, we propose to use the UCB method to consider interesting areas more frequently, for searching the optima. We partition the search space using a regular grid with M subdivisions along each dimension, resulting in $N = M^D$ areas. These areas are then considered as arms in a MAB problem, which let us use UCB to efficiently select the area of a new starting point.

This kind of approach proved to be efficient for numerical integration problems [17]. For MMO, its implementation is more complicated since it combines a restart strategy (using UCB) for dealing with the multiple optima, and a local search (with decreasing step-size) for estimating the rewards. Here, we propose the Algorithm 3, called Ucb Random-restarts with Decreasing Step-size (URDS).

Algorithm 3: URDS

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

1 **begin**
2 $\quad\hat{\mathbf{X}} \leftarrow \varnothing$
3 $\quad k \leftarrow 0$
4 $\quad$ { initialization step }
5 $\quad$ **foreach** *j* **do**
6 $\quad\quad k \leftarrow k + 1$
7 $\quad\quad \mathbf{x} \leftarrow \text{SampleArea}(j)$
8 $\quad\quad \hat{\mathbf{X}}, y \leftarrow \text{SearchDS}(f, \sigma_0, \epsilon_\sigma, y^*, \epsilon_y, \mathbf{x}, \epsilon_x, \hat{\mathbf{X}})$
9 $\quad\quad S[j] \leftarrow \frac{1}{y^* - y + 0.1}$
10 $\quad\quad A[j] \leftarrow 1$
11 $\quad$ { UCB step }
12 $\quad$ **while** *all optima not found* **do**
13 $\quad\quad k \leftarrow k + 1$
14 $\quad\quad j^* \leftarrow \arg\max_j \frac{S[j]}{A[j]} + R\sqrt{\frac{\ln k}{A[j]}}$
15 $\quad\quad \mathbf{x} \leftarrow \text{SampleArea}(j^*)$
16 $\quad\quad \hat{\mathbf{X}}', y \leftarrow \text{SearchDS}(f, \sigma_0, \epsilon_\sigma, y^*, \epsilon_y, \mathbf{x}, \epsilon_x, \hat{\mathbf{X}})$
17 $\quad\quad$ **if** $\hat{\mathbf{X}}' \neq \hat{\mathbf{X}}$ **then**
18 $\quad\quad\quad \hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}}'$
19 $\quad\quad\quad S[j^*] \leftarrow S[j^*] + \frac{1}{y^* - y + 0.1}$
20 $\quad\quad A[j^*] \leftarrow A[j^*] + 1$

The URDS algorithm is basically composed of two steps: the initialization step and the UCB step. The first step computes a local search in each of the N areas, for initializing the reward estimations: the initial rewards are stored in S and the numbers of selections are stored in A. The optimum found during this step are also stored, in $\hat{\mathbf{X}}$. Then, the second step is the restart strategy. It selects the most interesting area, using the UCB method (see equation 2), samples a new starting point in this area, computes a local search from this point and finally updates the corresponding reward (S and A). The UCB step terminates when all the optima are found ($\hat{\mathbf{X}}$).

The reward is defined as $\frac{1}{y^* - y + 0.1}$. The idea behind this formula is to give higher rewards to the points which are close to the real optimum y^* of the function. The algorithm can be modified quite easily for handling the case where y^* is unknown. For example, we can define the reward as the fitness of the optimum found, i.e. replace, in Algorithm 3, line 9 by $S[j] \leftarrow y$ and line 19 by $S[j^*] \leftarrow S[j^*] + y$.

The URDS algorithm requires setting two parameters. The parameter R is very classic in UCB methods; it tunes the trade-off between exploitation and exploration: a small value favors exploitation, a high value favors exploration. The parameter M is the number of subdivisions per dimension and gives the total number of areas to consider ($N = M^D$); it should be high enough to match the shape of the function but small enough to limit memory usage and to avoid a huge number of evaluations for initializing the estimations of the rewards.

V. Simulation results

A. Functions

To evaluate the proposed algorithm, we use several classic functions [9], [14]. We also use some modified functions which model more complex problems (see Fig. 1).

The function f_{Sin} is a multi-dimensional sine-based function $[0,1]^D \to [0,1]$ defined as:

$$f_{\text{Sin}}(\mathbf{x}) = \frac{1}{D}\sum_{d=1}^{D} \sin^{2s}(p\pi \mathbf{x}_d). \tag{3}$$

This function has two parameters: p is the number of peaks per dimension and s defines the sharpness of the peaks. This function is used in [9] and in [14], with $s = 3$ and $p = 5$. It has p^D optima which are regularly distributed (see Fig. 1a).

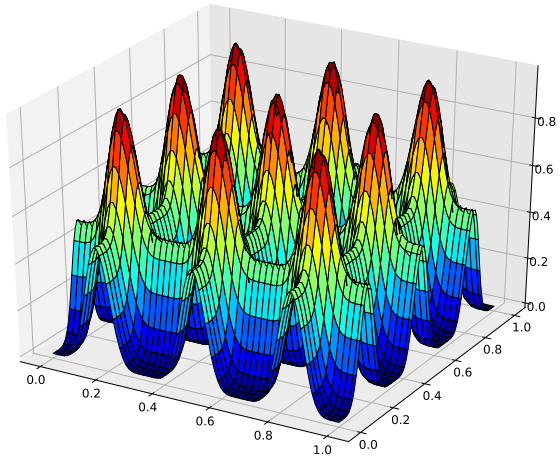
The function f_{SinBasin} is similar to f_{Sin} but with a large plateau (see Fig. 1b):

$$f_{\text{SinBasin}}(\mathbf{x}) = \begin{cases} f_{\text{Sin}}(\mathbf{x}) & \text{if } \|\mathbf{x}\|_\infty < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

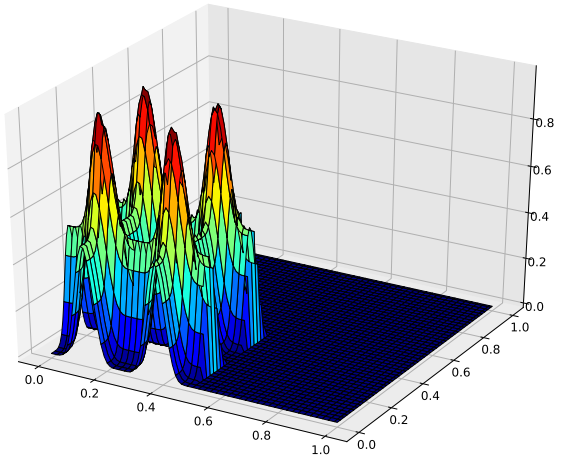
The function f_{Hump} is a simplified version of the Hump function presented in [9], where the random peaks have the same shape (see Fig. 1c):

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

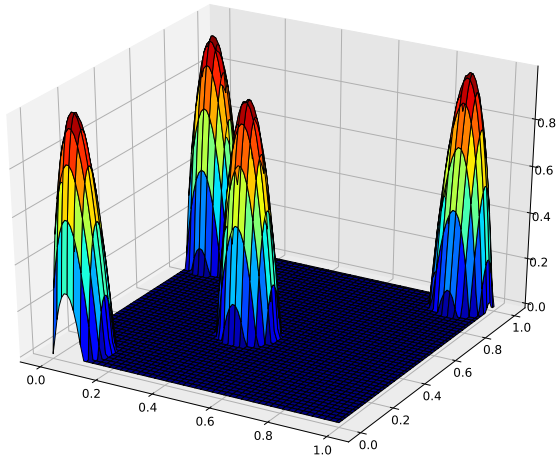
Here, α defines the (inverse) sharpness of the peaks, r the radius of the peaks and Q the number of peaks (and therefore the number of optima). The points $\mathbf{x_q}$ define the centers the peaks and are randomly drawn in $[0,1]^D$. This randomness



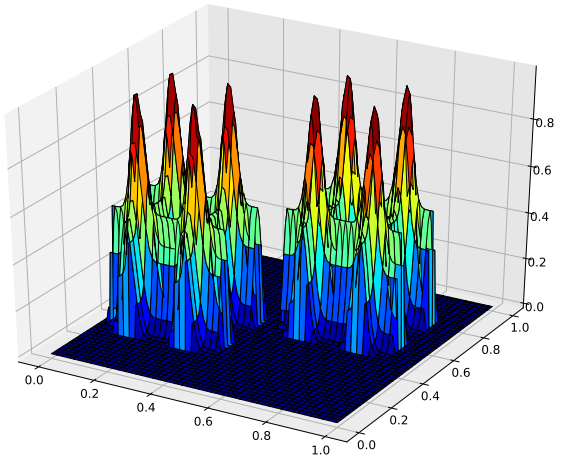
(a) f_{Sin} with $s = 3$ and $p = 3$ (see [9] and equation 3).



(b) f_{SinBasin} with $s = 3$ and $p = 4$ (see equation 4)



(c) f_{Hump} with $\alpha = 2$, $Q = 4$ and $r = 0.1$ (see [9] and equation 5).



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

Fig. 1: Multi-modal functions used in our experiments (represented in a 2D search space).

makes the function more difficult to optimize but the peaks are still locally simple.

Finally, the function f_{HumpSin} combines the randomness of f_{Hump} with a local complexity from f_{Sin} (see Fig. 1d):

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

Here, z is the number of zones and r the radius of the zones. The points $\mathbf{x_z}$ define the centers of the zones and are randomly drawn such that the zones do not overlap and are inside the search space. The number of optima is then $z \times p^D$.

B. Experiments and results

Using the previous functions, we compare the proposed algorithm (URDS) with the classic restart algorithms (RDS and QRDS) presented in Section II. We run each experiment 30 times and compute average values and 95% level confidence intervals.

In a first set of experiments, we measure the numbers of function evaluations that the algorithms need in order to find all the optima (see Table I). We notice that QRDS and URDS always perform better than RDS. This confirms the results found in [14], namely the superiority of QRDS over RDS. We also notice that QRDS and URDS have a quite similar performance on simple problems but URDS becomes better for higher dimension ($d = 3$ or $d = 5$) and for more complex functions (in particular f_{HumpSin} which is the most difficult function experimented here).

In a second set of experiments, we consider more complex problems and measure the number of optima found by the three algorithms after 10^6 evaluations (see Table II). URDS is clearly better than QRDS and RDS for the function f_{HumpSin}.

Function	D	RDS	QRDS	URDS	# Optima
f_{Sin} $[s=3,p=5]$	2	8723 ± 972	5361 ± 396	**5203** ± 438 $[R=100,M=5]$	25
f_{SinBasin} $[s=3,p=5]$		12251 ± 2306	5312 ± 639	**4675** ± 589 $[R=1,M=2]$	9
f_{Hump} $[\alpha=1,K=5,r=0.1]$		6296 ± 1399	**4067** ± 485	4663 ± 903 $[R=50,M=7]$	5
f_{HumpSin} $[s=4,p=8,z=2,r=0.1]$		235579 ± 17139	**164244** ± 6037	164340 ± 21293 $[R=2,M=3]$	128
f_{Sin} $[s=3,p=5]$	3	114298 ± 25678	**78124** ± 16430	81771 ± 13131 $[R=100,M=5]$	125
f_{Sin} $[s=3,p=6]$		206403 ± 21129	**143094** ± 12954	155076 ± 10033 $[R=100,M=9]$	216
f_{SinBasin} $[s=3,p=5]$		98940 ± 41683	59311 ± 12260	**43959** ± 16772 $[R=0.1,M=2]$	27
f_{SinBasin} $[s=3,p=6]$		70355 ± 8959	**57721** ± 7195	62957 ± 15797 $[R=5,M=7]$	27
f_{Hump} $[\alpha=1,K=5,r=0.1]$		81840 ± 15005	71421 ± 14414	**68762** ± 14225 $[R=100,M=5]$	5
f_{HumpSin} $[s=4,p=4,z=2,r=0.01]$		683312 ± 23255	561314 ± 21614	**458303** ± 43231 $[R=2,M=5]$	128
f_{SinBasin} $[s=3,p=4]$	5	359394 ± 25454	277184 ± 24792	**254704** ± 34479 $[R=2,M=3]$	32
f_{Hump} $[\alpha=1,K=5,r=0.1]$		54612 ± 7034	50451 ± 7455	**38566** ± 7390 $[R=2,M=3]$	5

TABLE I: Average number of evaluations needed by each algorithm to find all the optima (with 95% level confidence intervals).

This shows the relevance of the area selection implemented in URDS: the algorithm automatically learns the areas where the optima are concentrated. As expected, the three algorithms have quite similar results for f_{Sin}, since this function has regularly distributed optima.

In the two previous tables, the URDS results are the best results we obtain among the experimented parameter sets. The parameter R (trade-off between exploitation and exploration) is particularly interesting to consider, since we clearly see the influence of the function. Indeed, with functions where the optima are regularly distributed (such as f_{Sin} and f_{Hump} in low dimensions), the best results are obtained with a high value of R (exploration). With functions where the optima are concentrated in small areas (such as f_{SinBasin} and f_{HumpSin}), the best results are obtained with a small value of R (exploitation).

To illustrate the impact of the URDS parameters (R and M) more deeply, we plot the result of URDS for several functions and parameter sets (see Fig. 2). These plots show that the influence of R is related to the regularity of the distribution of the optima whereas M is related to the "shape" of non-regular distributions. For example, the f_{Sin} function presented in Fig. 2a has regularly distributed optima. Therefore, the best results are obtained for high R (exploration) and high M (which forces a regular distribution of the restart points). Similarly, the f_{Hump} function presented in Fig. 2b has also regularly distributed optima, hence the best results for high R and high M. The f_{HumpSin} function presented in Fig. 2c has its optima concentrated in two zones. The best results are thus obtained for low R (exploitation) and high M (which enables the algorithm to better fit these zones). Finally, the f_{SinBasin} function presented in Fig. 2d has regularly distributed optima but only in the first quadrant of the search space. Here, the most important parameter is M, since $M = 2$ perfectly fits the interesting area.

VI. Conclusion

In this paper, we propose a new restart strategy for multi-modal optimization. Our algorithm can handle complex functions, for example locally complex functions or functions where the optima are non-uniformly distributed over the search space. It partitions the search space and models the successive selections of a restart area as a multi-armed bandit problem, solved using the UCB method. This enables the algorithm to restart the local search in interesting areas.

Our algorithm outperforms classic restart-based algorithms, such as the quasi-random restarts with decreasing step-size algorithm, especially when the function to optimize is complex (locally complex, basins…).

As a drawback, this approach is limited to moderate dimensions since the regular partition of the search space makes the number of areas grows exponentially with the dimension. As a perspective, we would consider using a dimension-independent partition scheme.

Another drawback is the necessity to tune the two parameters of the URDS algorithm. As another perspective, we would remove the R parameter by using the UCB-TUNED method proposed in [18].

Acknowledgment

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).

References

[1] I. Rechenberg, "Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution," 1973.

[2] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[3] D. E. Goldberg, J. Richardson *et al.*, "Genetic algorithms with sharing for multimodal function optimization," in *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 1987, pp. 41–49.

[4] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems." Ph.D. dissertation, University of Michigan, 1975.

[5] S. W. Mahfoud, "Niching methods for genetic algorithms," *Urbana*, vol. 51, no. 95001, pp. 62–94, 1995.

[6] O. J. Mengshoel and D. E. Goldberg, "Probabilistic crowding: Deterministic crowding with probabilistic replacement," in *Proc. of GECCO conference*, 1999, pp. 409–416.

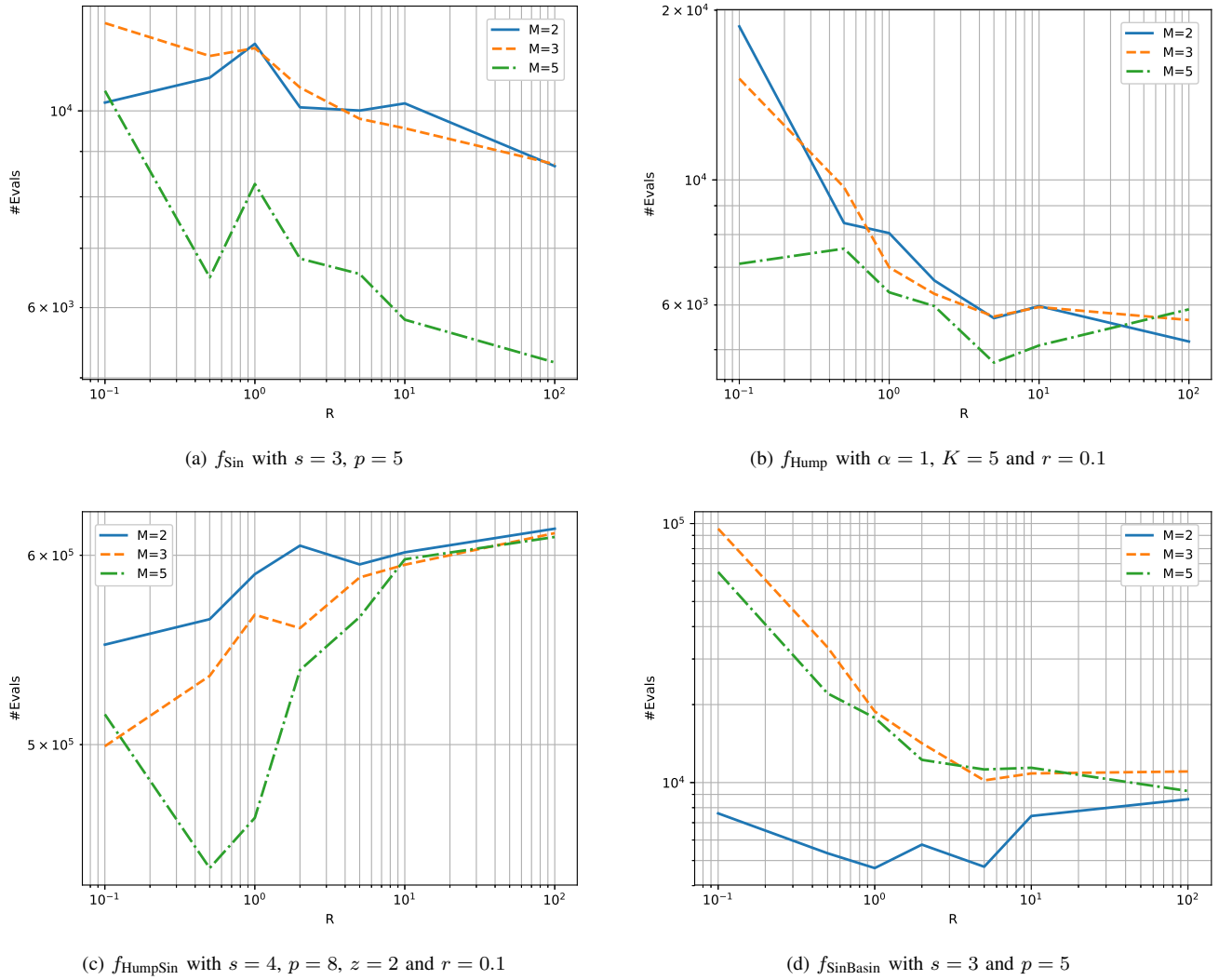
[7] X. Yin and N. Germay, "A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization," in *Artificial neural nets and genetic algorithms*, 1993, pp. 450–457.

[8] A. Pétrowski, "A clearing procedure as a niching method for genetic algorithms," in *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, 1996, pp. 798–803.

[9] G. Singh and K. Deb Dr, "Comparison of multi-modal optimization algorithms based on evolutionary algorithms," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1305–1312.

Function	D	RDS	QRDS	URDS	# Optima
f_{HumpSin} $[s=4, p=4, z=2, r=0.01]$	2	0.84%	95.31%	**99.68%** $[R=0.1, M=5]$	32
f_{HumpSin} $[s=4, p=8, z=2, r=0.1]$	3	66.7%	72%	**89.7%** $[R=2, M=5]$	1024
f_{Sin} $[s=3, p=4]$	5	91.6%	**96.15%**	92.49% $[R=100, M=3]$	1024
f_{HumpSin} $[s=4, p=8, z=2, r=0.1]$		5.6 10^{-4}%	5.5 10^{-4}%	**2.2 10^{-2}%** $[R=0.1, M=3]$	65536
f_{HumpSin} $[s=4, p=4, z=2, r=0.1]$		1.74%	1.73%	**28.9%** $[R=0.1, M=3]$	2048

TABLE II: Proportion of optima found after 10^6 evaluations (with 95% level confidence intervals).



(a) f_{Sin} with $s=3$, $p=5$

(b) f_{Hump} with $\alpha=1$, $K=5$ and $r=0.1$

(c) f_{HumpSin} with $s=4$, $p=8$, $z=2$ and $r=0.1$

(d) f_{SinBasin} with $s=3$ and $p=5$

Fig. 2: Impact of the parameters of the URDS algorithm, with $D=2$. R is the trade-off between exploitation (low R) and exploration (high R). M is the number of subdivisions per dimension.

[10] A. Auger and N. Hansen, "A restart cma evolution strategy with increasing population size," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 2, 2005, pp. 1769–1776.

[11] S. Kadioglu, M. Sellmann, and M. Wagner, "Learning a reactive restart strategy to improve stochastic search," in *International Conference on Learning and Intelligent Optimization*. Springer, 2017, pp. 109–123.

[12] F. Teytaud and O. Teytaud, "Qr mutations improve many evolution strategies: A lot on highly multimodal problems," in *Proc. of the 2016 GECCO conference*. ACM, 2016, pp. 35–36.

[13] A. Ahrari, K. Deb, and M. Preuss, "Multimodal optimization by covariance matrix self-adaptation evolution strategy with repelling subpopulations," *Evolutionary computation*, vol. 25, no. 3, pp. 439–471, 2017.

[14] M. Schoenauer, F. Teytaud, and O. Teytaud, "A Rigorous Runtime Analysis for Quasi-Random Restarts and Decreasing Stepsize," in *Artificial Evolution*, Angers, France, 2011.

[15] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in applied mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

[16] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, 2002.

[17] F. Leprêtre, F. Teytaud, and J. Dehos, "Multi-armed bandit for stratified sampling: Application to numerical integration," in *TAAI 2017*, Taipei, Taiwan, 2017.

[18] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.