On the huge benefit of quasi-random mutations for multimodal optimization with application to grid-based tuning of neurocontrollers

G. Chaslot[1] and J.-B. Hoock [2] and F. Teytaud[2] and O. Teytaud[2]

1- University of Maastricht

2- Tao, Inria, UMR Cnrs 8623, Lri, Bât. 490,
Université Paris-Sud, Orsay F-91405

Abstract.
In this paper, we study the optimization of a neural network used for controlling a Monte-Carlo Tree Search (MCTS/UCT) algorithm. The main results are: (i) the specification of a new multimodal benchmark function; this function has been defined in particular in agreement with [1] which has pointed out that most multimodal functions are not satisfactory for some real-world multimodal scenarios (section 2); (ii) experimentation of Evolution Strategies on this new multimodal benchmark function, showing the great efficiency of quasi-random mutations in this framework (section 3); (iii) the proof-of-concept of the application of ES for grid-based tuning Neural Networks for controlling MCTS/UCT (see section 3).

1 Introduction

We introduce in this section (i) neural nets (ii) evolution strategies in the parallel multimodal case (iii) quasi-random mutations (iv) Monte-Carlo Tree Search.

Neural Networks (NN). NN naturally lead to multimodal functions; even when they are used as a regression tool, they are highly multimodal, containing provably both plateaus and local minima [2, 3]. In the case of NN used as controllers of a dynamical system, the situation is worse: local minima can appear even with very simple NN due to the dynamics of the plant.

Evolution strategies (ES) and large population size for multimodal problems. ES [4, 5] are a well known framework for difficult optimization; in particular, multimodal and parallel optimization. In the related framework of Particle Swarm Optimization, [6] uses population-based optimization in order to train NN; a large value of the population size λ reduces the risk of local minima. This idea of λ large for reducing the risk of local minima is also classical in population-based optimization, far from NN [7, 8]. Unfortunately, it has been shown in [9] that some classical ES are not very efficient when λ is large. However, [9] also shew that the self-adaptive approach SA is quite stable and fast in that case, with a good speed-up as a function of λ. We therefore use the SA approach here.

There are not so many multimodal functions in classical benchmarks. If we consider e.g. Rastrigin's function, it is multimodal but an initialization large around the optimum leads to a far less "multimodal" run: quickly, the population

is in the right basin of attraction and the multimodality has no impact. [1] provides an interesting fitness function, which is designed in order to have a clear compromise between the size of basins of attraction - they show that most algorithms will focus on the largest basin which is not necessarily the best one (see also [10], emphasizing that in some robust optimization scenarios choosing the largest basin might be a good idea whenever the other basin leads to better fitness values). We define a highly multimodal fitness function in section 2.

Quasi-random (QR) mutations. After astonishing results in numerical integration [11], QR points have been used in several works around evolution strategies, in the initialization or in the mutations [12, 13]. [14] in particular pointed out that "modern" QR points [15], using e.g. scrambling, are far more efficient for mutations than old-fashioned QR points. [16] has presented positive results for QR mutations; in particular, [16] concludes to the efficiency of QR mutations in several settings. However, their observed improvement is moderate. We here show that QR mutations are much more efficient than that, in highly multimodal cases. The trick for generating a quasi-random Gaussian mutation is here as follows (similar to [14, 16]): there is only one quasi-random sequence (here Sobol's sequence [17]) in dimension N for an optimization problem in dimension N, and a Gaussian mutation is generated by applying the reverse of the cumulative normal distribution to the quasi-random vector (see [14], available freely on `http://hal.inria.fr`, for details; due to length constraints we can not give full details here).

An application: Monte-Carlo Tree Search (MCTS). MCTS [18, 19, 20, 21], including UCT[22], is a family of algorithms for taking decisions under uncertainty. It is known as particularly efficient for the game of Go, a very old Asian game in which computers are still far from the level of strong human players (whereas in chess, computers can win against humans whilst given them advantages). In particular, our program MoGo, based on MCTS, realized (i) the first ever win against a professional player in 9x9 Go (ii) the first ever win against a professional player in non-blitz 9x9 Go (Paris, 2008) (iii) the first ever win against a top level professional player in 19x19 Go (with handicap 9 however!). These results are based on increasingly complicated formula for the so-called "bandit" algorithm; the exact specification is the result of many complicated improvements, leading to the idea of using a NN instead of handcrafted formula. The schema is roughly as explained below. The principle of the algorithm consists in simulating thousands (or even millions in the parallel versions) of games starting at the current situation. The main point is that simulations are adaptive: based on previous simulations, simulations become more and more realistic and efficient. The algorithm follows:

while Time left > 0 **do**
  Initialize s to the current state of the goban.
  while s is not terminal **do**
    Compute the score of each legal move (see text) in s based on the situations/statistics stored in memory.
    Simulate the move with highest score; let s be the new state.

end while
Store the result of the game and the visited situations in a hashmap.
end while
Choose the most simulated move from the current situation.

The difficult part is the score (however, other parts above have been simplified for the sake of clarity; see papers cited above for more details). In a given situation, the score combines $\hat{p}(d)$ (online score, as in [22]), $\widehat{\hat{p}}(move)$ (the transient score, see [21]), $H(d)$ some heuristic value based on patterns(as in [19]).

$$Score(d) = \alpha \underbrace{\hat{p}(d)}_{Onlinevalue} + \beta \underbrace{\widehat{\hat{p}}(move)}_{Transientvalue} + \gamma \underbrace{H(d)}_{Offlinevalue} + \delta P(d,n) \quad (1)$$

where the coefficients α, β, γ and δ are empirically tuned coefficients depending on $n(d)$ (number of simulations of the decision d) and n (number of simulations of the current board) with the following constraints: (1) $\alpha + \beta + \gamma \simeq 1$ (not exactly equality, as seemingly a number > 1 is somewhat better for small values of $n(d)$); (2) $\alpha \simeq 0$ and $\beta \simeq 0$, i.e. $\gamma \simeq 1$, for $n(d) = 0$; (3) $\beta >> \alpha$ for $n(d)$ small; (4) $\alpha \simeq 1$ for $n(d) \to \infty$; (5) δ only depends on n and decreases to 0 as $n \to \infty$; (6) $d \mapsto P(d,n)$ converges to the constant function as $n \to \infty$; $\delta P(d,n)$ is analogous to the progressive unpruning term[19]. These rules imply that: (i) initially, the most important part is the offline learning; (ii) later, the most important part is the transient learning (RAVE values [21]); (iii) eventually, only the "real" statistics (online values) matter. In our experiments, we add a NN, which acts as a multiplicative term specifying α, β and γ (two coefficients are chosen by the NN, and these two terms are used as multiplicative factors in the formulas specifying these three variables).

Notations and methodological choices. Our real-world implementations are based on grids. This implies tricks for fault-tolerance and we therefore prefer very simple approaches in which we can easily catch errors and relaunch the algorithm when some jobs are lost. In particular we do not discuss niching [23, 24]; such approaches are interesting in the multimodal case and could be considered also, but we preferred the most simple tools. The improvements presented in this paper, based on QR mutations, are anyway somewhat orthogonal to niching mechanisms and niching-based algorithms could benefit from these improvements as well. In all the paper $[[a,b]]$ stands for $\{a, a+1, a+2, \ldots, b\}$.

2 A multimodal benchmark

[1] pointed out that there are some weaknesses in usual multimodal fitness functions. They propose a multimodal fitness function in which the number of basins can stay bounded independently of the dimension, and the size of the basins is controlled. We propose a highly multimodal function: at all scales, the fitness function is multimodal. One can consider that this fitness function is far too difficult for real-world applications; we agree that most real fitness functions are not so difficult. However, our goal is that, even if, involuntarily, you get rid of

dangerous local minima by tuning your algorithm in repeated experiments, you will get stuck in the next local minimum unless your algorithm is really strong for avoiding local minima. Interestingly, the positive results of [16] about QR mutations become much stronger in this setting.

The MM function, mapping $x \in \mathbb{R}$ to $MM_K(x) = step(x) + mod(step(x), K)$, for some parameter $K \in \{1, 2, 3, \dots\}$, where $step(x) = round(\log(x))$ where $round(x) = \lceil x + 0.5 \rceil$ and $mod(a, b) = r$ if $b|(a-r)$ and $r \in [[0, b-1]]$. In dimension $N > 1$, $MM_K(x) = \sum_{i=1}^{N} MM_k(x_i)$. This MM function is monomodal for $K = 1$, but with many plateaus; with $K > 1$ the function becomes more and more difficult, with local minima at all scales (`http://www.lri.fr/~teytaud/qrlllong.pdf` for Figures).

3 Experimental results

Below, we compare QR and random mutations on artificial benchmarks. We conclude to the very high efficiency of QR mutations for highly multimodal fitness functions. Then, we will apply QR mutation on a real world application involving NN and MCTS. In all this section, we use the SA algorithm, as in [9], due to its success in the parallel setting, parameterized as follows: (i) isotropic mutation (i.e. mutations are of the form $\sigma \times \mathcal{N}$ with $\mathcal{N}$ a standard multivariate Gaussian random variable) (ii) $\tau \simeq 1/\sqrt{N}$ (exact formula depending on the run, see details below) where N is the dimension (iii) initial parent $(1/\sqrt{N}, \dots, 1/\sqrt{N})$ for artificial experiments (not centered on the optimum) (iv) initial $\sigma = 1$.

Artificial experiments. We present in Fig. 1 normalized convergence rates for various values of λ, in (i) the classical random case and (ii) the QR case. The normalized convergence rate is evaluated when the fitness $-100N$ is reached.

Results with higher dimensionality or other values of τ are presented in `http://www.lri.fr/~teytaud/qrlllong.pdf`; we can essentially see that the efficiency of QR (compared to standard random numbers) holds on a bigger range of values of λ.

Real world experiments. The elements to be controlled are specified in section 1. The form of the controller is simply two neurons, with respectively 3 inputs and 1 bias / 2 inputs and 1 bias, leading to 7 parameters (there are also coefficients chosen by the human expert involved). The NN chooses some coefficients involved in the computation of α, β,γ and δ as a function of more statistics than in usual MCTS (see 1); more details can be provided by email on request. The objective function is the success rate in games against a baseline, as estimated over 200 games. There were 4 generations of 92 individuals; the small number of generations is due to the huge computational cost: each individual (evaluated in parallel) is evaluated in 5 or 6 hours. The average score for these generations were respectively 83.32, 84.10, 85.48, 90.68 (standard deviation 0.74%). The score of the mean individual of the last generation was $101.73 \pm 0.74\%$. As the score of the handcrafted version is 100.0, we have a moderate improvement; yet, there is a significant advantage in front of the initial distribution given to the ES, equivalent to a hand-tuning.

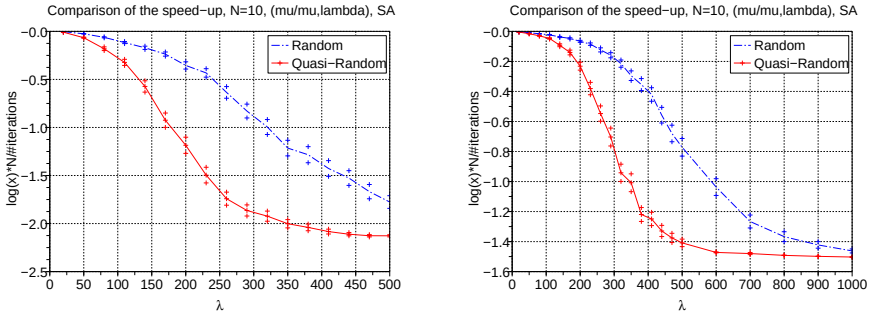

Fig. 1: We here present results in dimension 10 with $\tau = 1/\sqrt{N}$: left, $K = 1$; right, $K = 2$. Abscissa: λ, i.e. population size. Ordinate: the log of the distance to the optimum, multiplied by the dimension and divided by the number of iterations. The QR points have a huge improvement over the random points for a large range of values of λ; with $K = 1$, the QR version is 2.5 times faster for small values of λ, almost 4 times faster for $\lambda = 170$, and remains higher than 1.5 times faster for $\lambda = 350$. With $K = 2$, the QR version is 4.5 times faster for $\lambda = 230$, and remains at least 1.5 faster for $\lambda \leq 500$.

4 Discussion

On the methodological side, the results of this paper are twofolds: we propose a new highly multimodal function, and show the huge benefit of QR mutations on this benchmark. The results are in particular highly relevant for λ large, as for parallel optimization (as in our application). On the application side, we show how a NN tuned by a simple but well-chosen algorithm can lead to significant improvements in the difficult case of MCTS. In many MCTS, the heart of the system is unreadable; therefore, plugging and optimizing a NN is reasonable; automatic tuning as in this paper, with self-adaptive parallel ES, leads to results equivalent to tedious handcrafting, and the resulting code is not more unreadable.

References

[1] Monte Lunacek, Darrell Whitley, and Andrew Sutton. The impact of global structure on search. In Günter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *PPSN*, volume 5199 of *Lecture Notes in Computer Science*, pages 498–507. Springer, 2008.

[2] K. Fukumizu and S. Amari. Local minima and plateaus in hierarchical structures of multilayer perceptrons. *Neural Networks*, 13(3):317–327, 2000.

[3] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, November 1995.

[4] I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.

[5] H.-G. Beyer. *The Theory of Evolutions Strategies*. Springer, Heidelberg, 2001.

[6] F. van den Bergh and A. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers, 2000.

[7] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In X. Yao et al., editors, *Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of *LNCS*, pages 282–291. Springer, 2004.

[8] A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005*, pages 1769–1776, 2005.

[9] Hans-Georg Beyer and Bernhard Sendhoff. Covariance matrix adaptation revisited - the CMSA evolution strategy. In Günter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *Proceedings of PPSN*, pages 123–132, 2008.

[10] Kenneth A. De Jong. Genetic algorithms are not function optimizers. In L. Darrell Whitley, editor, *FOGA*, pages 5–17. Morgan Kaufmann, 1992.

[11] A. B. Owen. Quasi-Monte Carlo sampling. In H. W. Jensen, editor, *Monte Carlo Ray Tracing: Siggraph 2003 Course 44*, pages 69–88. SIGGRAPH, 2003.

[12] Shuhei Kimura and Koki Matsumura. Genetic algorithms using low-discrepancy sequences. In *GECCO*, pages 1341–1346, 2005.

[13] A. Auger, M. Jebalia, and O. Teytaud. Xse: quasi-random mutations for evolution strategies. In *Proceedings of Evolutionary Algorithms, 12 pages*, 2005.

[14] Olivier Teytaud and Sylvain Gelly. Dcma: yet another derandomization in covariance-matrix-adaptation. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 955–963, New York, NY, USA, 2007. ACM.

[15] P. L'Ecuyer and C. Lemieux. *Recent Advances in Randomized Quasi-Monte Carlo Methods*, pages 419 – 474. Kluwer Academic, 2002.

[16] Olivier Teytaud. When does quasi-random work?. In Günter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *PPSN*, volume 5199 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2008.

[17] I. M. Sobol. On the systematic search in a hypercube. *Siam journal on Numerical Analysis*, 16(5):790–793, October 1979.

[18] Rémi 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.

[19] G. Chaslot, M.H.M. Winands, J.W.H.M. 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.

[20] Yizao Wang and Sylvain 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.

[21] Sylvain Gelly and David 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.

[22] L. Kocsis and C. Szepesvari. Bandit-based monte-carlo planning. In *ECML'06*, pages 282–293, 2006.

[23] Ofer M. Shir and Thomas Bäck. Performance analysis of niching algorithms based on derandomized-es variants. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 705–712, New York, NY, USA, 2007. ACM.

[24] Stefan Bird and Xiaodong Li. Adaptively choosing niching parameters in a pso. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 3–10, New York, NY, USA, 2006. ACM.