Quasi-random numbers improve the CMA-ES on the BBOB testbed

Olivier Teytaud

Quasi-random numbers improve the CMA-ES on the BBOB testbed

Olivier Teytaud

TAO (Inria), LRI, UMR 8623 (CNRS - Univ. Paris-Sud),
bat 490 Univ. Paris-Sud 91405 Orsay, France, teytaud@lri.fr

Abstract. Pseudo-random numbers are usually a good enough approximation of random numbers in evolutionary algorithms. But quasi-random numbers follow a different idea, namely they are aimed at being more regularly distributed than random points. It has been pointed out in earlier papers that quasi-random points provide a significant improvement in evolutionary optimization. In this paper, we experiment quasi-random mutations on a well known test case, namely the Coco/Bbob test case. We also include experiments on translated or rescaled versions of BBOB, on which we get similar improvements.

1 Introduction

Monte Carlo is a classical method for computing approximate integrals. They can also be used directly for optimization; this is the simple random search algorithm. Evolutionary algorithms can be viewed as an improved form of random search, adaptively modifying the probability distribution in order to focus on the optimum. While Monte Carlo integration has been upgraded to Quasi Monte Carlo (also known as quasi-random), most evolution strategies use pseudo-random numbers, aimed at approximating random numbers, and not Quasi Monte Carlo, in spite of a few promising works in that direction. This might be due to lack of extensive experimental results on some classical testbeds; the purpose of this paper is to do this extensive experiment of quasi-random mutations in the Bbob/Coco benchmark.

In this paper we recall the state of the art in the use of quasi Monte Carlo in evolution strategies (Section 2), and then experiment an existing quasi Monte Carlo evolutionary algorithm on the Bbob/Coco framework.

2 Derandomization in evolution strategies

Evolution strategies[1] have been "derandomized" in several manners: use of covariance matrix[2, 3], and use of quasi-random points. We here consider the latter. It can be considered independently of the first and we will indeed use perform experiments in an algorithm which includes covariance matrix adaptation.

Low-dispersion or quasi-random points have been used for derandomizing the random search[4–6], or evolutionary algorithms[7] or other randomized optimization algorithms[8]. We here refer mainly to [9, 10], using quasi-random points

for derandomizing the mutations in the CMA-ES algorithm[11]. The quasi-randomized version of CMA is termed DCMA, which stands for derandomized-CMA.

Some important elements about quasi-random points follow. Computational cost is not a good reason for discarding quasi-random sequences. The computational cost for generating quasi-random points is negligible and indeed often smaller than for classical pseudo-random numbers[12, 13]. Quasi-random sequences are different from pseudo-random sequences. Quasi-random numbers are not a special case of pseudo-random numbers. Pseudo-random sequences are aimed at imitating random sequences, whereas quasi-random sequences are aimed at doing better, thanks to a better uniformity. Additionally, modern quasi-random sequences have a random part[14]. Quasi-random points have low discrepancy, decreasing as the inverse of the number of points (within logarithmic factors), whereas pseudo-random numbers and random numbers, by design, have discrepancy decreasing as the inverse of the square root of the number of points. Pseudo-random numbers are an approximation of random numbers, whereas quasi-random numbers are qualitatively different. The weaknesses of old quasi-random sequences (such as non-scrambled Halton sequences), which were often worse than random sequences in high dimension, have been overcome thanks to randomized quasi-random sequences; these sequences have the good properties of quasi-Monte Carlo methods and are at least as performant as Monte Carlo methods in most (if not all) cases[15–19].

3 Experimental results

We follow the experimental setup proposed in "exampleexperiment.m" provided in the Bbob/Coco downloads; a comment in the file states that the number of function evaluations should be increased, so we increase to $100 \times D$ with D the dimension for the strict Bbob/Coco setting in Section 3.1, which will be extended to $2000D$ in Section 3.3. We will also check translated or rescaled versions of Bbob. All experiments are performed with initial point $(0, 0, \ldots, 0)$ and initial step-size 1. The version of CMA-ES is the Matlab/Octave one as of the time of submission. All quasi-random numbers are obtained by the scrambled Halton method.

3.1 Experimental results in the Bbob/Coco setting

In this section, we produce results using the Bbob/Coco framework, without any change except the increase of the number of evaluations to $100 \times D$ (we increased this because it is recommended in the Bbob/Coco sample file to do so). The Bbob/Coco framework has been used in several conferences.Results are presented in Fig. 7 (frequency of success depending on the number of evaluations, for different precision levels). Fig. 1 presents the scatter plots, i.e. the x-axis is the computation time for reaching some precision for the default CMA whereas the y-axis is the computation time for reaching the same precision for DCMA. All

graphs are obtained by Bbob/Coco automatically, so that there is no parameter choice by ourselves. All experiments use BBOB V13.09.

3.2 Experiments in the parallel setting

We reproduce the results above in the parallel setting. We will assume here that we consider a problem in which the computational cost is mainly in the fitness evaluations, and that function evaluations have an approximately constant computational cost, so that increasing the population size is a natural solution for parallelization: the population size is the number of processors.

We set the population size to $20 \times D$, where D is the dimension, and do not modify anything else in the Bbob/Coco framework. Results are presented in Fig. 2 (frequency of approximate solving on the y-axis for the number of evaluation given on the x-axis).

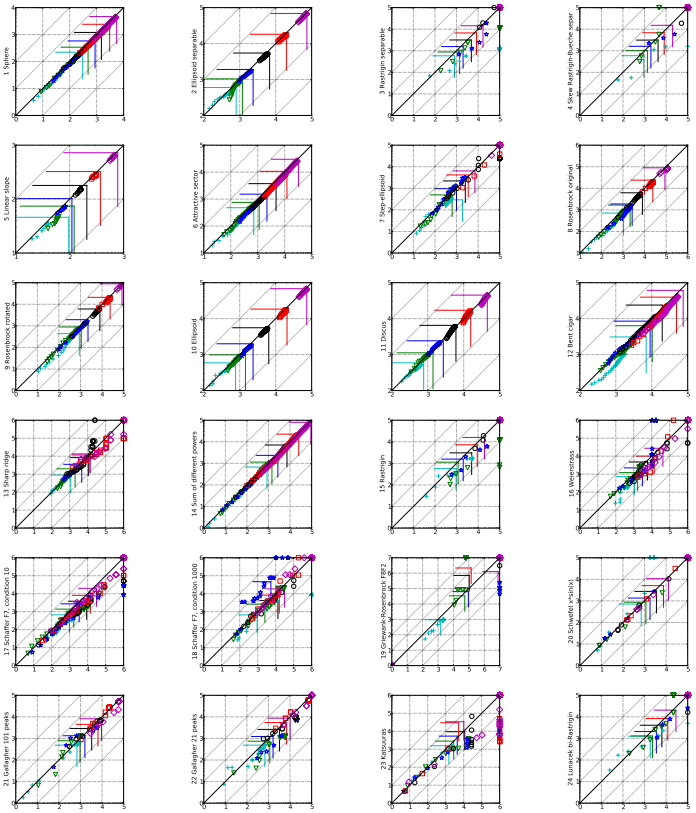

Fig. 1. Experimental results (scatterplots) for the default Bbob framework. For each graph, corresponding to one function from f1 to f24 in Bbob/Coco, the x-axis is the run length for the default CMA in log-10 scale, whereas the y-axis is the run length in log-10 scale for the quasi-randomized version, i.e. DCMA. CMA is better than DCMA for function f18, in the sense that there are more points above the diagonal than below. DCMA is better for the 22 other functions.

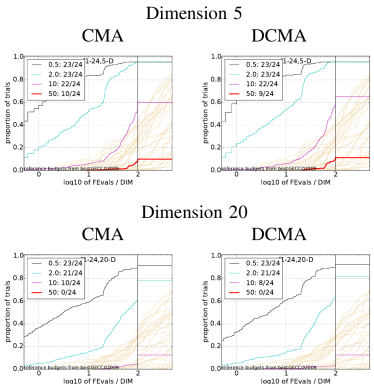

Fig. 2. Experiments on Bbob with population size forced to a larger value $20 \times D$ where D is the dimension. Success rates for different number of function evaluations as in Fig. 7. Left: results with the default CMA. Right: results with quasi-randomization (DCMA). Results are usually better for DCMA, but the difference is smaller than with the standard population size of CMA.

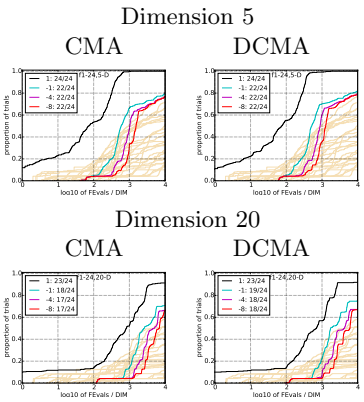

Fig. 4. Results in the parallel setting (20D as population) and with larger numbers of function evaluations ($10000D$). Left: results with the default Cma. Right: results with quasi-randomization. Results similar to the standard case.

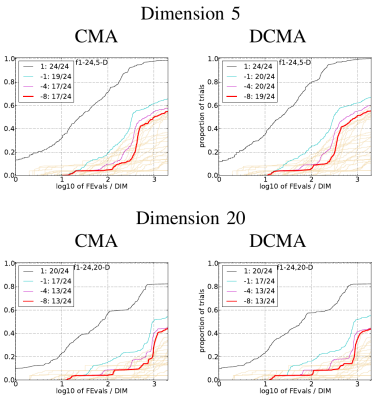

Fig. 3. Results in the original Bbob setting but with larger numbers ($2000D$) of function evaluations. Left: results with the default Cma. Right: results with quasi-randomization. The difference is smaller than in the other cases.

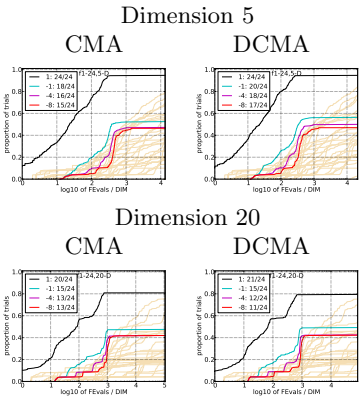

Fig. 5. Comparison between CMA and DCMA on the rescaled testbed. The difference between CMA and DCMA is similar to the difference in the original BBOB testbed; CMA outperforms DCMA on f19.

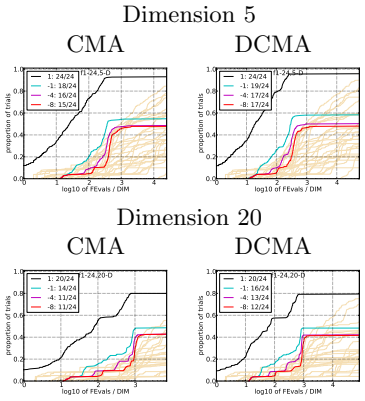

Fig. 6. Comparison between CMA and DCMA on the translated testbed. In dimension 5 DCMA outperforms CMA, but in dimension 20 it is the case only for curves 1, and -1; for -8 CMA outperforms DCMA and for -4 it is the same. The higher the better.

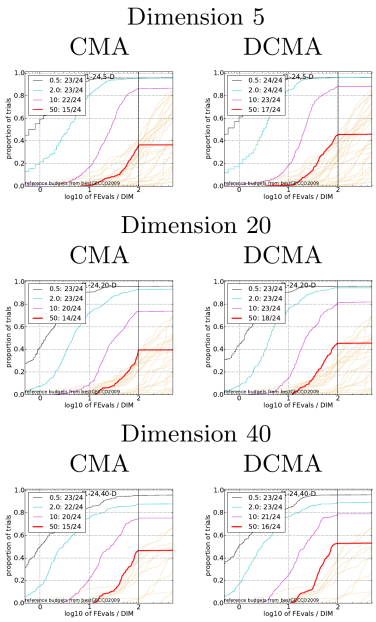

Fig. 7. Experimental results (percentage of success for different numbers of function evaluations; each curve corresponds to a different success criterion in terms of simple regret) for the default Bbob framework. Left: results with the default Cma. Right: results with DCMA. DCMA is usually faster. Fig. 1 presents the same results as scatter plots.

3.3 Experiments with larger numbers of iterations

We come back to the original Bbob/Coco setting of Section 3.1, but with $2000 \times D$ function evaluations in dimension D. Results are presented in Fig. 3 and still show a superiority of DCMA but with a smaller difference. Detailed results show a strong superiority for f12, f15, f16, f17, f18, f19, f23, f24.

3.4 Experiments with large population size and large numbers of iterations

We come back to the Bbob/Coco setting of Section 3.2, i.e. population size equal to $20D$ where D is the dimension, but with $10000 \times D$ function evaluations in dimension D. Results are presented in Fig. 4 and Fig. 8 and still show a superiority of DCMA, though not for all functions.

4 Experiments with modified BBOB

In this section, we rescale the BBOB testbed. As in the original experiments (Section 3.1), we use $100D$ function evaluations. Instead of working on $f(x)$, we work on $f(x/1000)$. Results are presented in Fig. 5 and 9. The superiority of DCMA over CMA is bigger, suggesting that derandomized mutations improve the robustness w.r.t an imperfect initialization (guessing the initial step-size is not that easy in real situations) leads to a roughly linear landscape.

5 Experiments with another modified BBOB

In this section, we translate the BBOB testbed. As in the original experiments (Section 3.1), we use $100D$ function evaluations. Instead of working on $f(x)$, we work on $f(x + 7)$ ($+7$ is added coordinate-wise, i.e. all d decision variables are shifted in dimension d). Results are presented in Fig. 6 and 10. The improvement by DCMA over CMA is bigger than in the original BBOB.

6 Conclusion

The derandomization proposed in [10] basically works. There are settings in which the difference is large, and settings in which the effect of quasi-randomization is minor; but it is rarely detrimental. The contribution of this paper are (i) confirming this superiority on the BBOB testbed (ii) efficiency of DCMA compared to CMA is preserved with large population sizes (iii) it is preserved in all Bbob dimensions (v) we confirm that the improvement is better in multimodal settings; this is consistent with [20].

We perturbated the BBOB testcase, just by changing the scale by a factor 1000, or by translating functions by $+7$. Results are essentially preserved.BBOB does not provide confidence intervals. This is deeply rooted in BBOB: there is

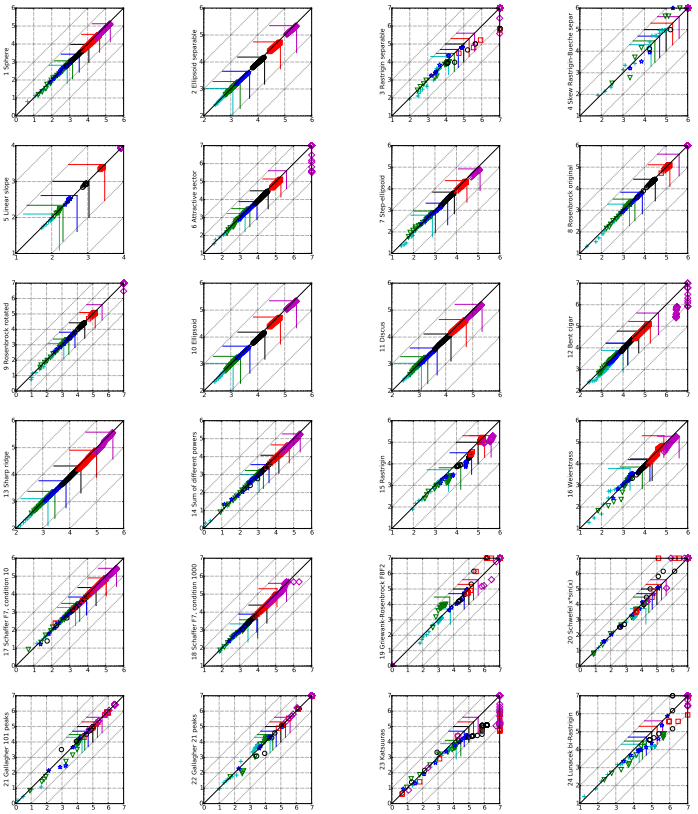

Fig. 8. Scatterplots in the parallel setting as in Section 3.2 (population size $20D$) but with larger numbers of function evaluations ($10000D$); text in Section 3.4. For each graph, corresponding to functions f1 to f24 in the Bbob/Coco framework, the x-axis is the run length for the default CMA in log-10 scale, whereas the y-axis is the run length in log-10 scale for the quasi-randomized version. DCMA outperforms CMA in the sense that there are more points below the diagonal than above for most functions, but the difference is often small; the difference is bigger for f3, f6, f12, f15, f16, f23, f24. CMA outperforms DCMA for f4 and f20.

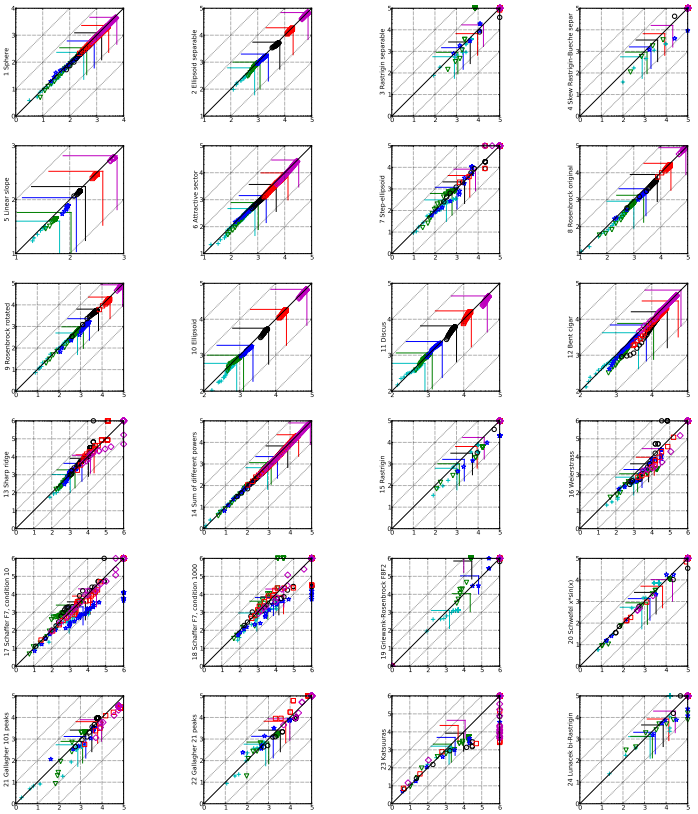

Fig. 9. Comparison between CMA and DCMA on the rescaled testbed. The difference between CMA and DCMA is similar to the difference in the original BBOB testbed. The improvement is visible on nearly all functions except f19 in the sense that there are more points below than above the curve. The difference is clearer on multimodal functions.

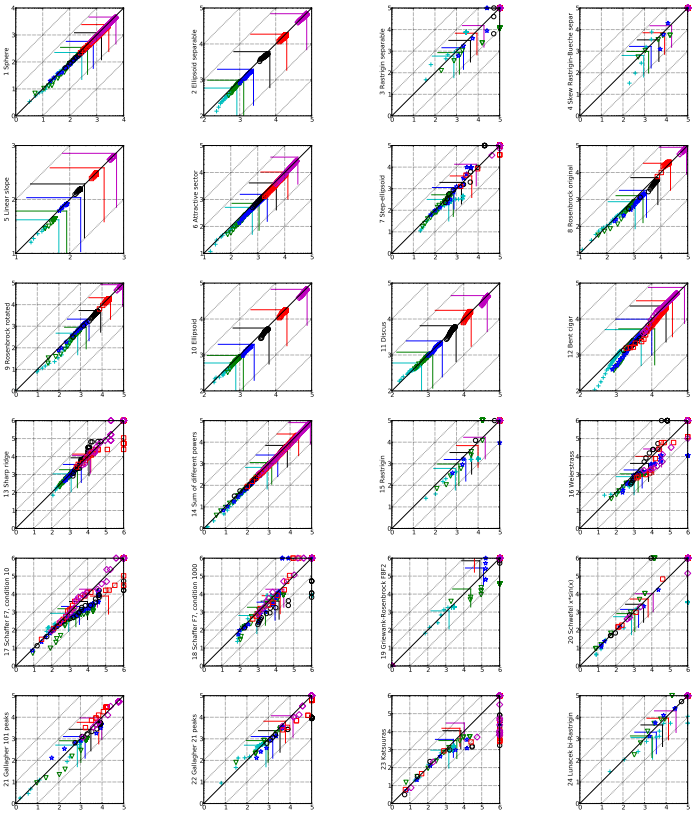

Fig. 10. Comparison between CMA and DCMA on the translated testbed. Each graph represents a function. X-axis = number of function evaluations for reaching the target precision for CMA. Y-axis = number of function evaluations for reaching the target precision for DCMA. The difference between CMA and DCMA is similar to the difference in the original BBOB testbed. The improvement is visible on nearly all functions (in the sense that we have more points below than above the curve), in particular multimodal.

a finite set of functions, and therefore overfitting is always possible, a trivial algorithm successively sampling the finite set of optima of BBOB instances for the considered dimension would have excellent performance. Nonetheless, we reproduced the results many times, and always got the same result, including translations and rescaling. All tested frameworks have been presented.

We considered results with respect to the number of fitness evaluations, not computation time; this is the standard Coco/Bbob methodology. The computational cost of the quasi-random part is negligible, indeed the computational complexity of quasi-random numbers is often less than the one of pseudo-random numbers. We decided to run experiments on the Bbob/Coco framework without any adaptation so that at least the framework is not chosen specifically for the experiments and results are neutral. There was no tuning at all and presented results are the results of the first set of runs in each setting.

We now discussion limitations of the present paper. In the present work, we just validated the derandomization of mutations by quasi-random numbers. Other derandomizations, based on symmetries as the one proposed in [21], might provide additional improvements; these two derandomizations can be combined. In the present paper we combine quasi-Monte Carlo and Covariance Matrix Adaptation, we could have symmetrized sampling combined with quasi-Monte Carlo and Covariance Matrix Adaptation, all together. Our experiments are performed with the scrambled Halton sequence. We do not claim that other, in particular older Quasi-Monte Carlo sequences would be as efficient. It is well known that old Quasi Monte Carlo sequences were not that good, in particular in high dimension[18]. There are now many good quasi-random sequences in the literature. Maybe other quasi-random sequences would provide better results.

The experiments were performed without any modification of CMA other than adding the quasi-random part, i.e. replacing $arz = random\ gaussian$ by $arz = quasi\ random\ gaussian$ (where arz is the notation in CMA for the mutation before rescaling and applying the covariance transformation). It is likely that the optimal parameters for the covariance update and for the step-size update are different from the optimal parameters for the original CMA. Therefore there is likely margin for improving the results of the DCMA algorithm, which is left as further work.

Quasi-random, or low-dispersion, can be used also for the restarts. This is the purpose of other published works. We did not include quasi-random restarts in order to separate both effects. Still, the performance improvement might be due to a better spreading of the initialization over the domain. We conjecture that the improvement related to quasi-random restarts will be larger than the one with quasi-random mutations - the purpose of this paper is basically that we can also include quasi-randomization in mutations.

References

1. H.-G. Beyer, *The Theory of Evolution Strategies*, ser. Natural Computing Series. Springer, Heideberg, 2001.

2. N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaption," in *Proc. of the IEEE Conference on Evolutionary Computation (CEC 1996)*. IEEE Press, 1996, pp. 312–317.
3. H.-G. Beyer and B. Sendhoff, "Covariance matrix adaptation revisited - the CMSA evolution strategy," in *Proceedings of PPSN*, G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, Eds., 2008, pp. 123–132.
4. H. Niederreiter, "Low-discrepancy and low-dispersion sequences," *J. Number Theory*, 1988.
5. ——, *Random Number Generation and Quasi-Monte-Carlo Methodes*. Society of Industrial and Applied Mathematics, 1992.
6. S. R. Lindemann and S. M. LaValle, "Incremental low-discrepancy lattice methods for motion planning," in *Proceedings IEEE International Conference on Robotics and Automation*, 2003, pp. 2920–2927.
7. S. Kimura and K. Matsumura, "Genetic algorithms using low-discrepancy sequences." in *GECCO*, 2005, pp. 1341–1346.
8. A. Georgieva and I. Jordanov, "A hybrid meta-heuristic for global optimisation using low-discrepancy sequences of points," *Computers and Operations Research, - special issue on hybrid metaheuristics*, In press.
9. O. Teytaud and S. Gelly, "DCMA, yet another derandomization in covariance-matrix-adaptation," in *GECCO*, D. Thierens et al. , Ed., London Royaume-Uni, 2007, pp. 955–922. [Online]. Available: http://hal.inria.fr/inria-00173207/en/
10. O. Teytaud, "When does quasi-random work?." in *PPSN*, ser. Lecture Notes in Computer Science, G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, Eds., vol. 5199. Springer, 2008, pp. 325–336. [Online]. Available: http://dblp.uni-trier.de/db/conf/ppsn/ppsn2008.html#Teytaud08
11. N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 11, no. 1, 2003.
12. T. Warnock, "Computational investigations of low-discrepancy point sets," in *In: S.K. Zaremba, Editor, Applications of Number Theory to Numerical Analysis (Proceedings of the Symposium), University of Montreal*, 1972, p. 319343.
13. ——, "Computational investigations of low-discrepancy point sets ii," in *In: H. Niederreiter and P.J.-S. Shiue, Editors, Monte-Carlo and Quasi-Monte-Carlo Methods in Scientific Computing, Springer, Berlin*, 1995.
14. M. Mascagni and H. Chi, "On the scrambled halton sequence," *Monte-Carlo Methods Appl.*, vol. 10, no. 3, pp. 435–442, 2004.
15. X. Wang and F. Hickernell, "Randomized halton sequences," *Math. Comput. Modelling*, vol. 32, pp. 887–899, 2000.
16. B. Tuffin, "A new permutation choice in halton sequences," *Monte-Carlo and Quasi-Monte-Carlo*, vol. 127, p. 427435, 1997.
17. I. M. Sobol, "On the systematic search in a hypercube," *Siam journal on Numerical Analysis*, vol. 16, no. 5, pp. 790–793, Oct. 1979.
18. A. Owen, "Multidimensional variation for quasi-Monte-Carlo," S. University, Ed., 2004.
19. B. Vandewoestyne and R. Cools, "Good permutations for deterministic scrambled halton sequences in terms of l2-discrepancy," *Computational and Applied Mathematics*, vol. 189, no. 1,2, p. 341:361, 2006.
20. G. Chaslot, J.-B. Hoock, F. Teytaud, and O. Teytaud, "On the huge benefit of quasi-random mutations for multimodal optimization with application to grid-based tuning of neurocontrollers." in *ESANN*, 2009. [Online]. Available: http://dblp.uni-trier.de/db/conf/esann/esann2009.html#ChaslotHTT09

21. S. Gelly, J. Mary, and O. Teytaud, "On the ultimate convergence rates for isotropic algorithms and the best choices among various forms of isotropy," in *Parallel Problem Solving from Nature-PPSN IX*. Springer, 2006.