Benchmarking the CMA-ES with Rank One Update on the bbob Noiseless Test Suite

Raneem Madani
Master 2 Optimization
University Paris-Saclay
Orsay, France
raneemmadani7@gmail.com

Ikhlas Enaieh
institutionMaster 2 Optimization
University Paris-Saclay
Orsay, France
ikhlasinayeh15@gmail.com

Zeidan Braik
Master 2 Optimization
University Paris-Saclay
Orsay, France
zaidane.1999.1@gmail.com

ABSTRACT

This paper examines the rank one update of the CMA-ES algorithm and compares its performance using the noiseless suite of the BBOB (Black-Box Optimization Benchmarking) benchmark using the COCO platform. The maximum number of function evaluations per run is set to 10^5 times the dimension of the search space, with the default population size, 30 restarts and no increasing population. And 10^4 for a large population size of 300 and 30 restarts, and the population was increased by a factor of 2 after each restart. The algorithm did very well with low population, but on the other hand it had a bad performance with high population expect for f_5 all of the instance, target pairs were completely solved for all dimensions.

CCS CONCEPTS

•Computing methodologies → Continuous space search;

KEYWORDS

(BBOB) Black-Box Optimization Benchmarking, (ECDF) Empirical Cumulative Distribution Function, Expected Running Time.

ACM Reference format:
Raneem Madani, Ikhlas Enaieh, and Zeidan Braik. 2022. Benchmarking the CMA-ES with Rank One Update on the bbob Noiseless Test Suite. In *Proceedings of the Genetic and Evolutionary Computation Conference 2022, Boston, MA, USA, July 9–13, 2022 (GECCO '22),* 8 pages.
DOI: 10.1145/123_4

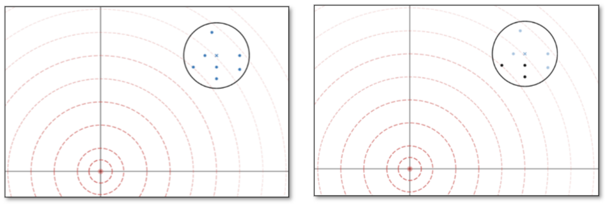
1 INTRODUCTION

The covariance matrix adaptation algorithm CMA-ES is an evolution strategy that adapts arbitrary, normal mutation distributions within a completely derandomized adaptation scheme with accumulation [3] and [8], and this is a stochastic derivative-free optimization algorithm. It is indeed one of the milestone algorithms used in optimizing an objective function $f(x) : \mathbb{R}^n \mapsto \mathbb{R}$ in a Black-box scenario where you neither know the function nor the gradient, but only the input and its respective output.

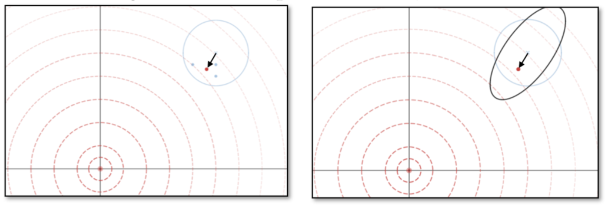
[*]Submission deadline: January 31st.

2 ALGORITHM PRESENTATION

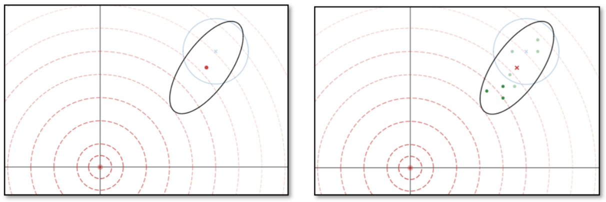
Now, like in References [7],[1] and [3], we will graphically depict the CMA-ES Algorithm with Rank one Update in a nutshell, by demonstrating how it works with an example function (Sphere in $\mathbb{R}^3$).



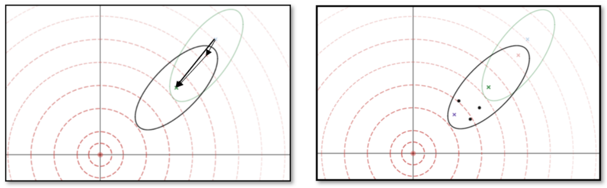
The CMA method generates multivariate Gaussian distributed samples $y_i \sim \mathcal{N}(0, C)$ with the following initial parameters: the mean m_t, the initial covariance matrix $C_{init} = Id$, and a step size σ_t. We start by drawing a sample λ , which is the population size (offspring), then we evaluate $X_i = m_t + \sigma y_i$, and finally we compute $f(X_i)$ and rank them. Now that we have decided the number of our candidate solutions (parents) μ, we select the best μ solutions in the ranking. (in our example take $\lambda = 7, \mu = 3$)



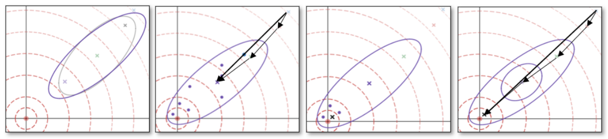
Then we compute $y_w = \sum_{i=1}^{\mu} w_i y_{i:\lambda}$, where $y_{i:\lambda}$ represents the ranked drawn samples corresponding to the ranked $f(Xi)$, and the mean update is: $m_{t+1} = m_t + \sigma_t y_w$. So y_i samples that you draw are the possible directions for your new mean and it's worth mentioning that we don't draw the $X_i \sim \mathcal{N}(m, \sigma^2 C)$. we just compute it as mentioned above. After we have updated the mean we will update the step size that decides the step length σ, the step size update is stated in [sec.4] [8] and consists of updating the evolutionary path p_σ using CSA [3] and then updating σ (at first, p_σ is set to 0).



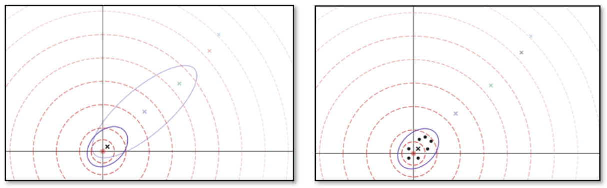
Now the critical part of our updating scheme is the covariance matrix update which distinguishes this algorithm. In the rank one update case, we first update the evolutionary path p_c according to the following $p_c = (1 - C_c)p_c + \sqrt{1 - (1 - C_c)^2}\sqrt{\mu_w}y_w$ where C_c is a constant strictly less than one, $\mu_w = \frac{1}{\sum w_i^2}$, $(1 - C_c)$ is the decay factor and $\sqrt{1 - (1 - C_c)^2}\sqrt{\mu_w}$ is a normalization factor [sec.4] [8]. The resultant vector p_c is important in reducing the number of function evaluations which will decrease the complexity as well, one can say that history information is accumulated in the evolution path. Now the updated covariance matrix C which will decide the shape of our new ellipsoid is $C_{t+1} = (1 - C_{cov})C_t + C_{cov}\, p_c\, p_c^T$ where C_{cov} is a constant strictly less than C_c and around $\frac{2}{n^2}$ and C_t is the old Covariance matrix. Then we move our new covariance matrix in the direction of y_w using the mean update and a step length equivalent to the updated step size.



We can see how p_c cumulate all the previous paths so the algorithm learns faster, and the process of drawing samples continues.



The algorithm keeps learning and moving towards the global minimum by successively updating the mean, step size, and covariance matrix.



Until it finally adapts the best covariance matrix with less axis ratio around the global minimum.
This is in a nutshell how the algorithm work, in the end, the following question is raised, why is it called rank one? Because the matrix's rank ($p_c\, p_c^T$) you use to update C is one because this matrix is generated by one vector only (p_c) and the whole updating process in each step is done towards that vector.

3 EXPERIMENTAL PROCEDURE

To Benchmark the CMA-ES Algorithm with rank one update two tests has been conducted:
First One:, Algorithm tested on BBOB noisless data with the following parameters: (Population Size default ($4 + 3\log(n)$) where n is the dimension, μ (the parents/ Candidate solutions) is default ($\lfloor \frac{\lambda}{2} \rfloor$), budget multiplier is 10^5, restarts 30 and incpopsize 1 so the population size will not increase after each restart.) here population is treated to be low.
Second One: Algorithm tested on BBOB noisless data with the following parameters: (Population Size is 300, μ (the parents/ Candidate solutions) is default ($\lfloor \frac{\lambda}{2} \rfloor$), budget multiplier is 10^4 and 30 restarts, the population size will by multiplied by a factor after every restart which we specified to be 2 using the argument: incpopsize) here population is treated to be High.

Then we compared the results of the low, high Outputs trials with the following four Algorithms:

- RANDOMSEARCH-5
- BFGS scipy 2019 Varelas: as it has available data for all the dimensions unlike the BFGS scipy Baudis that has no data for dimension 3 and 10
- BIPOP CMA ES hansen noiseless
- best 2009: which is an algorithm that takes the best parameters among a 31 algorithms.

Several things were put into account for the comparisons criteria, the single function performance among all the dimensions and how much it could solve of the instance target pairs with the given budget multiplier, correlation between function type and algorithm, the affects that population size play on the performance, which classes were perfectly solved and the Pros and Cons of the rank one update.

ECDF plots which are the output of COCO will be used for the above regards, we have 51 targets varying from 100 to 10^{-8} and 15 instances where the instances differs in the location of the local, global mimima and other parameters.

The functions in BBOB are divided to:

- Separable Functions ($f_1 - f_5$)
- Functions with Low or Moderate Conditioning ($f_6 - f_9$)
- Functions with High Conditioning and Uni-modal ($f_{10} - f_{14}$)
- Multi-modal Functions with Adequate Global Structures ($f_{15} - f_{19}$)
- Multi-Modal Functions with Weak Global Structure ($f_{20} - f_{24}$)

One missing trial was not conducted which is the performance with high population and low μ.

4 CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the CMA-ES with rank one update only with default population size $4 + 3\log(n)$ and no restarts on the entire bbob test suite [6] for $10^5 n$ function evaluations according to [9] and another trial have been conducted on a population size of 300 with 30 restarts and incpopsize equals to 2 on the entire bbob test suite [6] for $10^4 n$ function evaluations according to [9] .

The Code was run on AMD Ryzen 7 5800H with 8 cores on Python 3.10 in windows 11. The time per function evaluation for dimensions 2, 3, 5, 10, 20, 40 equals 8, 7.2, 6.6, 8.3, 6.6 and 7.6 times 10^{-5} seconds respectively for the default population and 5.6, 5.9, 5, 5.1, 5.3 and 6.1 times 10^{-5} for the large population.
its worth mentioning that μ for both experiments was default $\lfloor \frac{\lambda}{2} \rfloor$.

5 RESULTS

Results from experiments according to [9] and [4] on the benchmark functions given in [2] are presented in Figures 1, 2, 3, for low population and Figures 4, 5 and 6 for high population and Table 1 .

The experiments were performed with COCO [5], version 2.6, the plots were produced with version 2.6.

The **expected runtime (ERT)**, used in the figures depends on a given target precision, $I_{\mathrm{target}} = f_{\mathrm{opt}} + \Delta f$, and is computed over all relevant trials as the number of function evaluations executed during each trial while the best function value did not reach f_{t}, summed over all trials and divided by the number of trials that actually reached f_{t} [9, 10]. **Statistical significance** is tested with the rank-sum test for a given target Δf_{t} using, for each trial, either the number of needed function evaluations to reach f_{t} (inverted and multiplied by -1), or, if the target was not reached, the best Δf-value achieved, measured only up to the smallest number of overall function evaluations for any unsuccessful trial under consideration.

Results of Low Population Trial:

We First Present our results on the low population test: the instances target pairs were completely solved in ten functions (f_1, f_2, f_5, f_6, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{21}), but f_{24} was the only function that all it's instance target pairs were not met. f_{19} only converged in 2D, Uni-modal Functions on low dimensions were completely solved (i.e all the targets were satisfied) except for f_{13} as it is not smooth.

Compared to the other algorithms, on low dimension the performance of rank one was slightly same as the default CMA for all functions over all targets up to 5D, on the other hand, the performance for higher dimension was same to default CMA, up to 10^3 budget multiplier and then the default CMA enhanced much better and faster. Generally speaking rank one update did better than BFGS-Scipy and Randomsearch, its worth mentioning that the performance of rank one update was very close to Best 2009 for 2D, 3D and 5D.

One can say Uni-modal functions with low population all their instance target pairs were solved, and the separable were solved except for f_3, f_4 as they are multi-modal with 10^D local minimum.

Results of High Population Trial:

Now, we consider the High population trial, 21 functions their instance target pairs were not completely solved in any dimension, except for f_5 which is the linear slope function that was the only function to have its instance target pairs completely solved in all the dimensions with high population, f_1 Sphere function, it's instance target pairs were not solved in 2D with 2×10^4 and 3D with 3×10^4 functions evaluations it needed more to solve them all, while for 5D, 10D, 20D and 40D all the pairs were completely solved with the budget we have, because the sphere function has small condition number meaning that this is easier for optimization algorithm to converge to the global minimum however in low dimensions the search space is smaller and the optimization algorithm is more likely trapped in local minimum which are suboptimal solutions, therefore the reason why the sphere function converges in high dimensions but not in low dimension is due to the tradeoff between the difficulty of reaching the global minimum and the difficulty of avoiding local minimum. f_{17} in 2D, 3D, 5D and 10D the pairs were not completely solved with the budget as it is multimodal while it got completely solved in 20D and 40D as its condition number is low.

In high dimensions the optimization algorithm is more likely to reach the global optimum but more population size is needed.

The rank one update compared to other algorithms has a performance close to randomsearch in 2D which was really bad and BFGS Scipy was way better than it as many instance target pairs were solved in the scipy case, generally speaking around 40% of the instance target pairs were solved with rank one on all the functions in all the dimensions, BFGS scipy performance went down in larger dimensions and become closed to rank one and the gap between rank one and best 2009 was very large compared to low population.

High Conditioning functions even with being Uni-Modal (with one local which is a global) with Rank one not even 40% of the instance target pairs were solved, the performance was very low but on the other hand scipy was with a higher performance.

6 CONCLUSIONS

CMA-ES with rank one update provide a very high performance in low population and low dimension, but falls to the risk of stuck in a local minimum in multi-modal functions, rank one can learn the hessian matrix but don't take so much information from population that's why large population is not helpful, so it is better to use this algorithm when you have a unimodal function and low population and if you are minimizing in a dimension less than 5D with low population, rank one update works well.

REFERENCES

[1] Paul Dufossé, Asma Atamna, Philippe R Sampaio, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. 2022. Building scalable test problems for benchmarking constrained optimizers. *? ?* (2022), ?

[2] S. Finck, N. Hansen, R. Ros, and A. Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Presentation of the Noiseless Functions.* Technical Report 2009/20. Research Center PPE. http://coco.lri.fr/downloads/download15.03/bbobdocfunctions.pdf Updated February 2010.

[3] Nikolaus Hansen and Anne Auger. 2014. Principled Design of Continuous Stochastic Search: From Theory to Practice. In *Theory and Principled Methods for the Design of Metaheuristics*, Yossi Borenstein and Alberto Moraglio (Eds.). Springer, 145–180. https://hal.inria.fr/hal-00808450

[4] N. Hansen, A Auger, D. Brockhoff, D. Tušar, and T. Tušar. 2016. COCO: Performance Assessment. *ArXiv e-prints* arXiv:1605.03560 (2016).

[5] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. 2021. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *Optimization Methods and Software* 36 (2021), 114–144. Issue 1. DOI: http://dx.doi.org/https://doi.org/10.1080/10556788.2020.1808977

[6] N. Hansen, S. Finck, R. Ros, and A. Auger. 2009. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions.* Technical Report RR-6829. INRIA. http://hal.inria.fr/inria-00362633/en/

[7] N. Hansen and A. Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation.* 312–317. DOI: http://dx.doi.org/10.1109/ICEC.1996.542381

[8] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. DOI: http://dx.doi.org/10.1162/106365601750190398

[9] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. 2016. COCO: The Experimental Procedure. *ArXiv e-prints* arXiv:1603.08776 (2016).

[10] Kenneth Price. 1997. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation.* IEEE, Piscataway, NJ, USA, 153–157. DOI: http://dx.doi.org/10.1109/ICEC.1997.592287

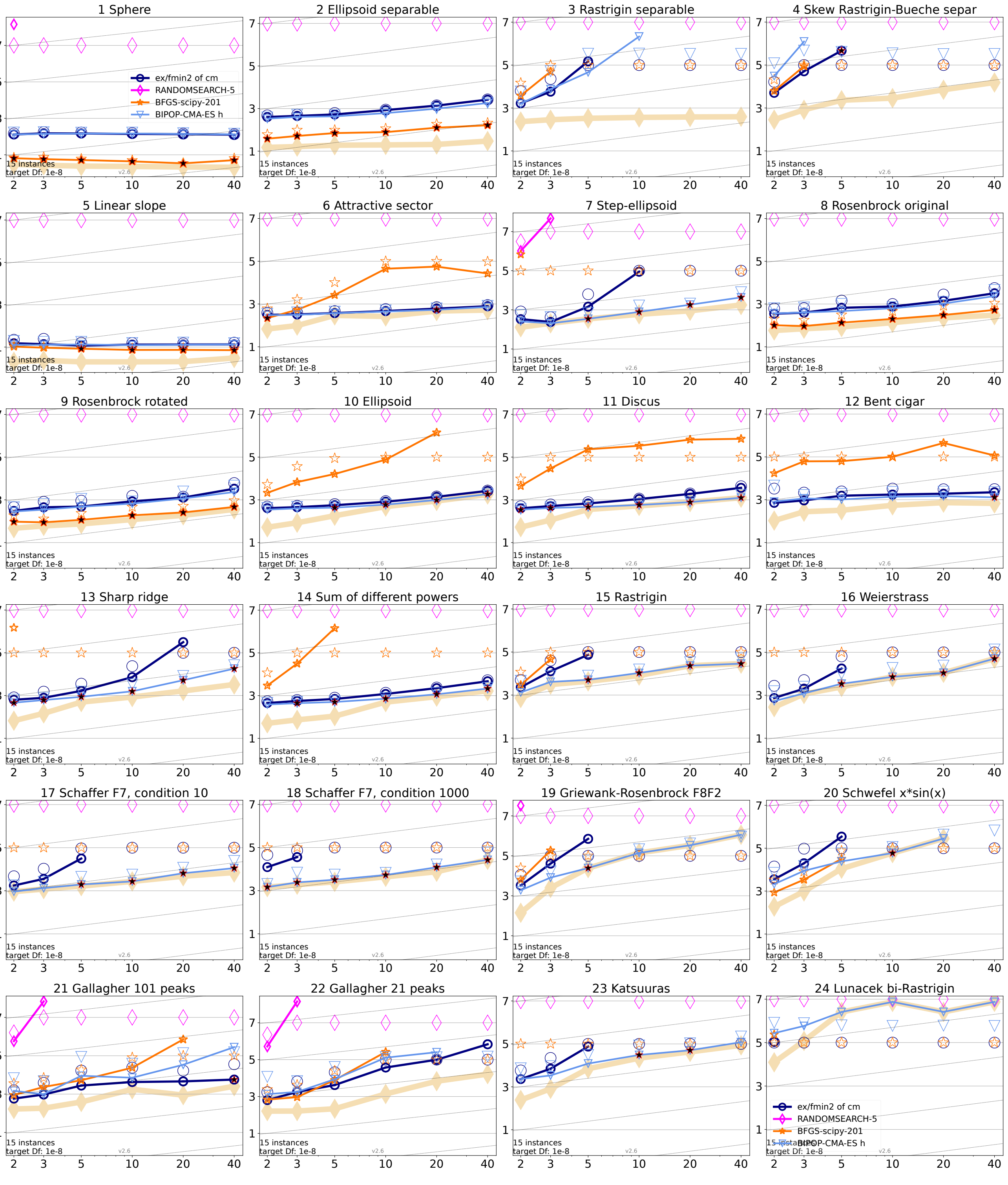


Figure 1: Expected running time (ERT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: BFGS-scipy-2019 Varelas, ◇: BIPOP-CMA-ES hansen noiseless, ⋆: RANDOMSEARCH-5-1e7D Brockhoff, ▽: ex/fmin2 of cma.evolution strategy 100000D on bbob.

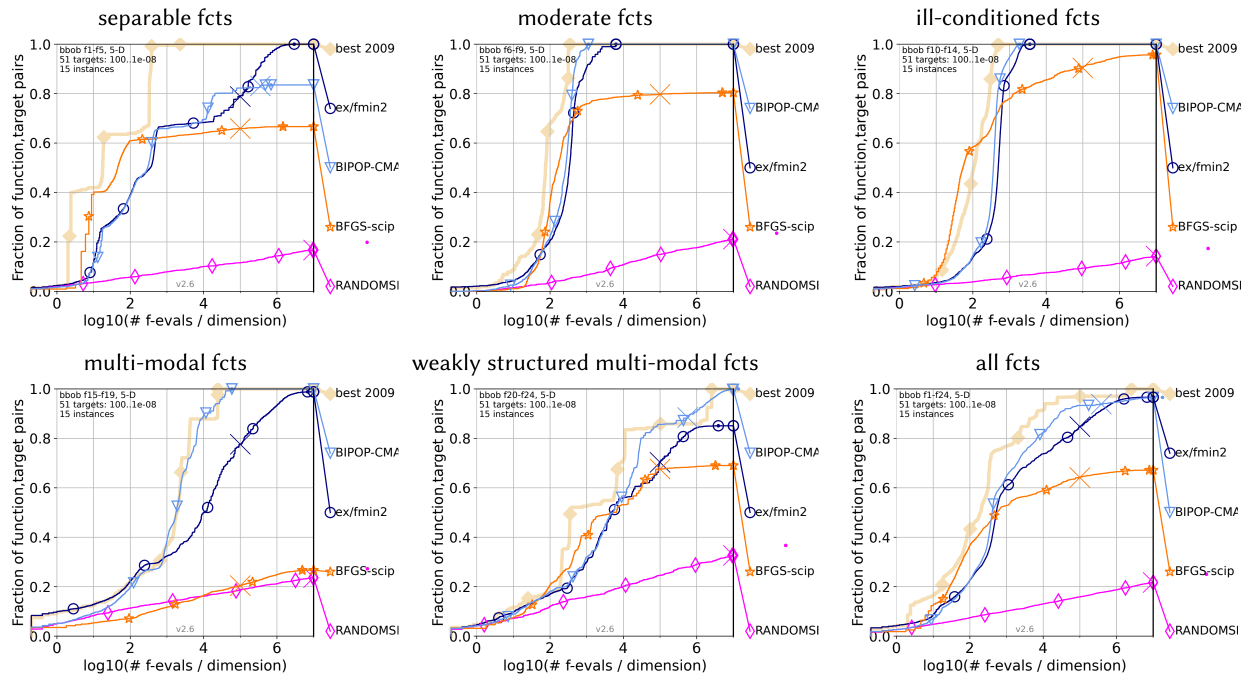


Figure 2: Bootstrapped empirical cumulative distribution of the number of f-evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

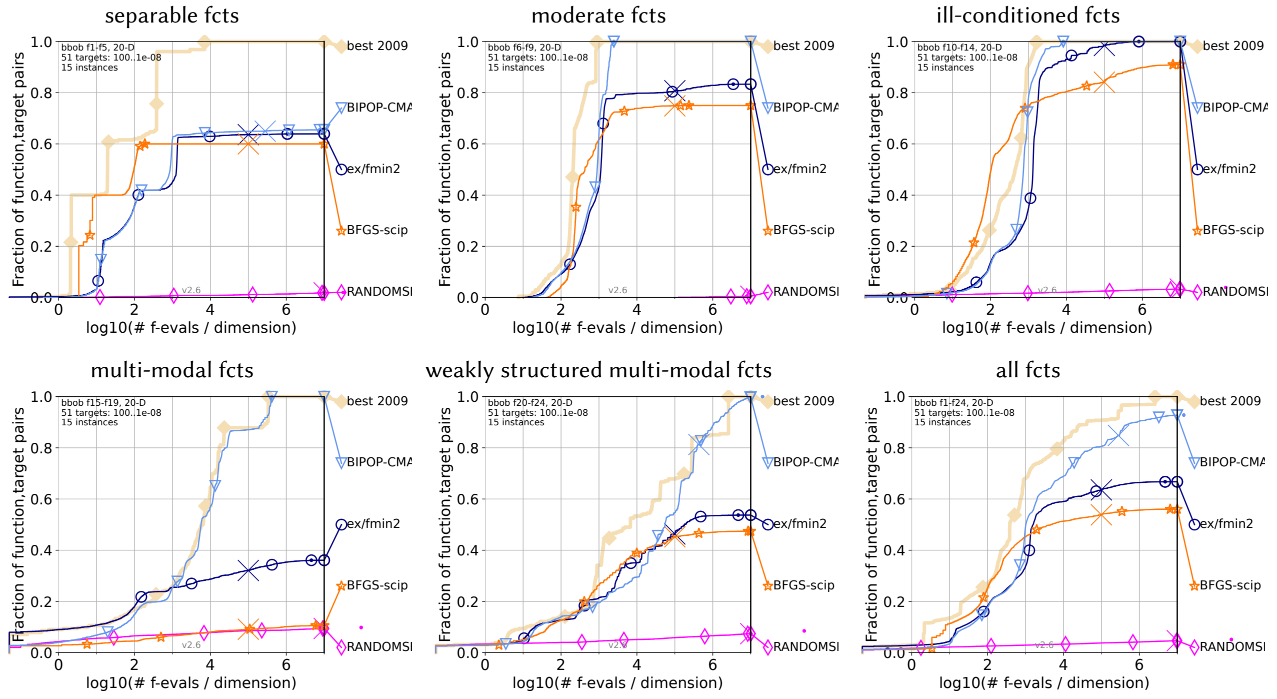


Figure 3: Bootstrapped empirical cumulative distribution of the number of f-evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 20-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

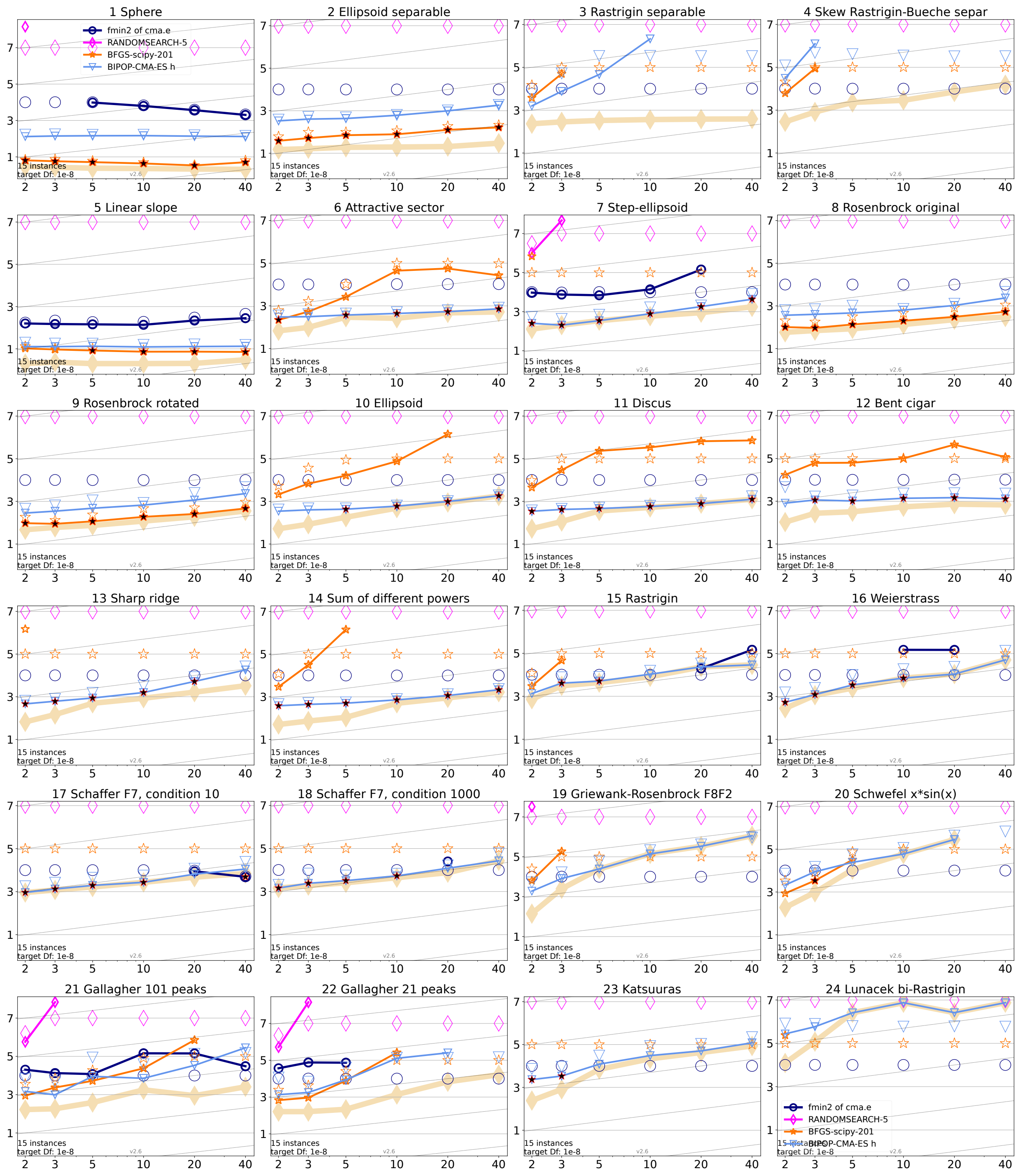


Figure 4: Expected running time (ERT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: BFGS-scipy-2019 Varelas, ◇: BIPOP-CMA-ES hansen noiseless, ⋆: RANDOMSEARCH-5-1e7D Brockhoff, ▽: ex/fmin2 of cma.evolution strategy 10000D on bbob.

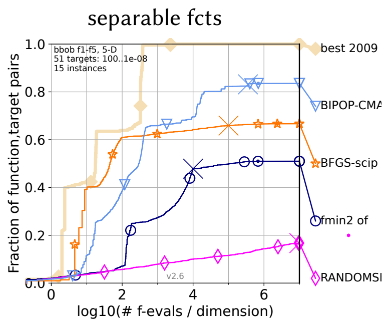
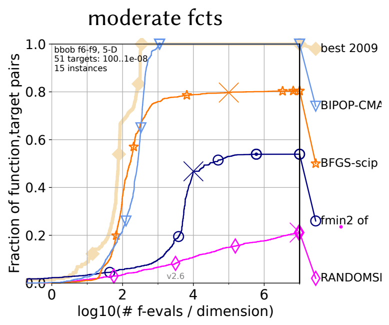
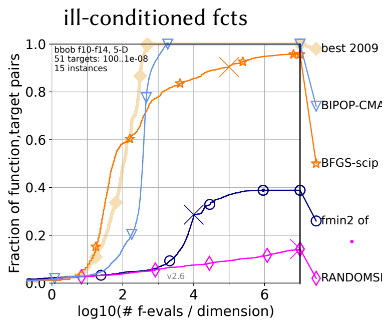
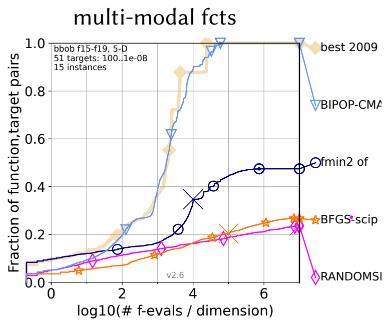
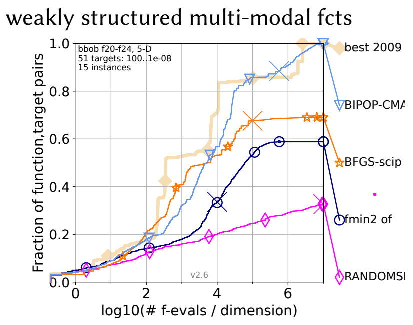
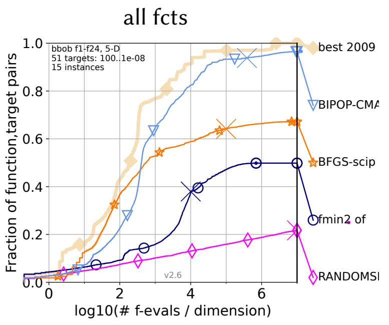







Figure 5: Bootstrapped empirical cumulative distribution of the number of f-evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

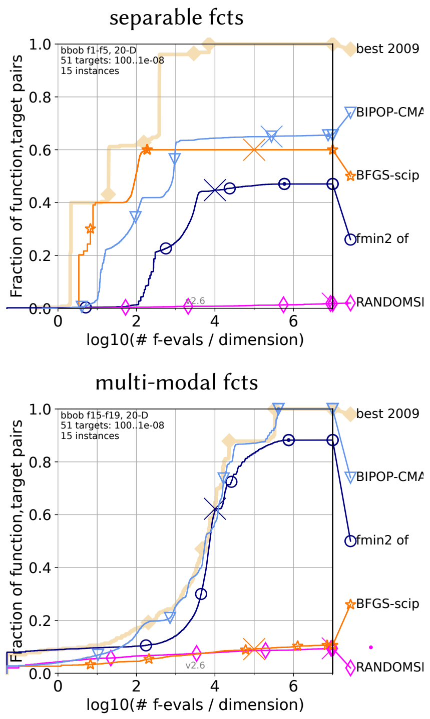
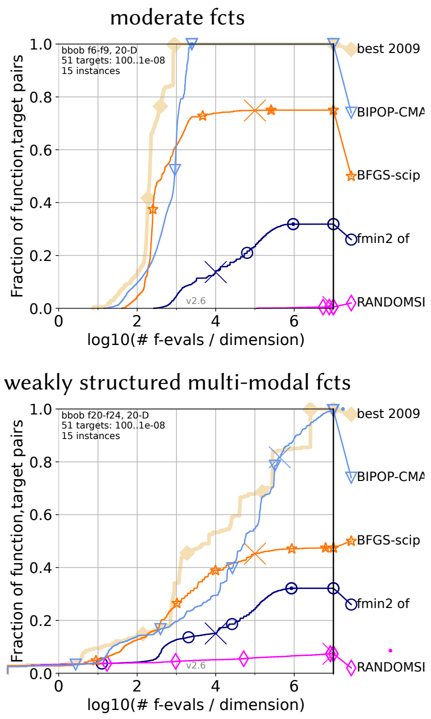
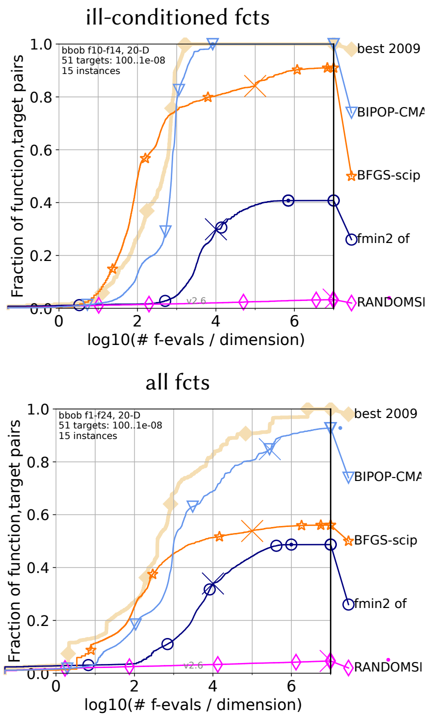




Figure 6: Bootstrapped empirical cumulative distribution of the number of f-evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 20-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

	2D		3D		5D		10D		20D		40D	
Function	Low	High	Low	High	Low	High	Low	High	Low	High	Low	High
f1	10^2	10^2	10^2	10^3	10^2	10^4	10^2	10^4	10^2	10^4	10^2	10^3
f2	10^3	25%	10^3	20%	10^3	10%	10^3	10%	10^3	3%	10^3	0%
f3	10^4	35%	10^4	25%	60%	18%	19%	17%	15%	15%	3%	10%
f4	10^4	25%	90%	19%	37%	15%	16%	18%	13%	10%	3%	5%
f5	10^1	10^2	10^1	10^2	10^1	10^2	10^1	10^2	10^1	10^3	10^1	10^3
f6	10^3	45%	10^3	40%	10^3	30%	10^3	30%	10^3	0%	10^3	0%
f7	10^3	10^4	10^3	10^4	10^4	95%	85%	95%	21%	30%	18%	17%
f8	10^3	50%	10^3	30%	10^3	25%	10^5	25%	10^4	17%	10^4	10%
f9	10^3	50%	10^3	30%	10^3	30%	10^5	30%	10^3	17%	10^4	10%
f10	10^3	30%	10^3	19%	10^3	10%	10^3	10%	10^3	2%	10^4	0%
f11	10^3	30%	10^3	19%	10^3	10%	10^3	10%	10^2	2%	10^4	0%
f12	10^4	20%	10^3	15%	10^4	20%	10^4	20%	10^2	41%	10^4	8%
f13	10^3	35%	10^3	35%	10^3	38%	10^4	37%	90%	39%	65%	37%
f14	10^3	65%	10^3	65%	10^3	63%	10^3	65%	10^3	62%	10^4	62%
f15	10^4	39%	10^5	30%	80%	25%	18%	25%	10%	58%	5%	21%
f16	10^3	42%	10^4	30%	10^5	25%	45%	25%	25%	32%	21%	37%
f17	10^4	42%	10^4	45%	10^5	50%	65%	50%	55%	10^4	75%	10^4
f18	10^5	38%	10^5	38%	65%	41%	50%	45%	40%	87%	30%	77%
f19	10^4	70%	90%	45%	45%	30%	30%	30%	25%	32%	27%	36%
f20	10^4	40%	10^5	50%	45%	25%	21%	25%	20%	20%	27%	36%
f21	10^3	85%	10^4	80%	10^4	77%	10^4	77%	10^4	22%	10^5	39%
f22	10^3	80%	10^4	38%	10^4	36%	90%	35%	65%	17%	30%	17%
f23	10^4	25%	10^4	25%	85%	21%	37%	21%	37%	19%	34%	18%
f24	75%	20%	30%	28%	20%	10%	17%	10%	10%	3%	5%	0%

Table 1: Percentage of (Instance, Target) pairs solved within the number of function evaluations per dimension on the bbob functions on the dimensions of 2,3,5,10,20 and 40, comparing high population (= 300) (H) with 10^4 budget multiplier and low population (default) (L) with 10^5 budget multiplier, where the percentages designate that not all the pairs were solved but it amounts to percentage of the pairs solved within the specified budget, and the values 10^n means that all the pairs for the respective function, dimension and population were solved but we show the real actual budget needed (i.e. the number of function evaluations needed = $D \times 10^n$ where D stands for the respective dimension).