Benchmarking the ATM Algorithm on the BBOB 2009 Noiseless Function Testbed

Benjamin Bodner
Brown University
Providence, Rhode Island
benjamin_bodner@brown.edu

ABSTRACT

In deep learning and physical science problems, there is a growing need for better optimization methods capable of working in very high dimensional settings. Though the use of approximated Hessians and co-variance matrices can accelerate the optimization process, these methods do not always scale well to high dimensional settings. In an attempt to meet these needs, in this paper, we propose an optimization method, called Adaptive Two Mode (ATM), that does not use any *D*x*D* objects, but rather relies on the interplay of isotropic and directional search modes. It can adapt to different optimization problems, by the use of an online parameter tuning scheme, that allocates more resources to better performing versions of the algorithm. To test the performance of this method, the Adaptive Two Mode algorithm was benchmarked on the noiseless BBOB-2009 testbed. Our results show that it is capable of solving 23/24 of the functions in 2D and can solve higher dimensional problems that do not require many changes in the direction of the search. However, it underperforms on problems in which the function to be minimized changes rapidly in non-separable directions, yet mildly in others.

CCS CONCEPTS

• **Computing methodologies → Continuous space search**;

KEYWORDS

Benchmarking, Black-box optimization

ACM Reference format:
Benjamin Bodner. 2019. Benchmarking the ATM Algorithm on the BBOB 2009 Noiseless Function Testbed. In *Proceedings of Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, July 13–17, 2019 (GECCO '19 Companion),* 8 pages.
https://doi.org/10.1145/3319619.3326802

1 INTRODUCTION

The Two Mode algorithm employs a combination of two search methods, an isotropic random search and a directional random search. These two modes complement each other by serving the roles of exploration (isotropic search) and exploitation (directional search). To switch between the two search strategies, we use a simple set of rules, which can be tuned in real-time.

2 ALGORITHM PRESENTATION

The Two Mode algorithm always starts out in the isotropic search mode, which randomly samples the local environment around the initial search point, using a uniform distribution. If no beneficial search direction is initially found, the algorithm invests more in exploration, by growing the radius of its isotropic search mode. Once a promising direction that leads to improvement is found that meets the set of switching rules, the algorithm causes the directional mode to grow exponentially, while causing the isotropic mode to rapidly decay. In order to satisfy our switching rules, the sampled direction must reduce the objective function and the L2 norm of the displacement from the initial point must be greater than an adaptive threshold. This exponential growth of the directional search allows the algorithm to quickly exploit promising search directions and find new local minima along them. Once the directional search does not find any new points that lead to improvement (either because the mode has grown too much or has reached a local minimum along its line), it begins to decay exponentially. As the directional mode decays (exploitation), the isotropic mode begins to grow again (exploration), in hope of finding an improved direction to exploit. The whole process repeats itself over and over until the algorithm finds a local minima that meets one of the stopping conditions. It is important to note that both of these modes have suppression mechanisms to constrain their growth to within the boundaries of the problem and prevent overflows. To avoid premature convergence in multimodal scenarios, the search is restarted if the algorithm fails to improve for 30 iterations.

Because this multimodal search strategy is quite general, the algorithmic details, such as the distributions used for the random search of the modes and the rate of growth of each mode, may be changed without impacting the overall philosophy. Instead of tuning the parameters manually, we employed online parameter search to find sets of parameters that work well within the local environments sampled by the algorithm. This is similar in essence to the approach taken in [8], but leverages parallelism, rather than sequential steps. This allows the algorithm to adapt to different situations in a way that increases its performance.

2.1 ONLINE PARAMETER TUNING

As most optimization algorithms rely on sets of parameters to govern the optimization process (such as CMA-ES in [10]), it is important that we find optimal sets to enhance the performance of our algorithm. Because the local structure of objective functions can change throughout different stages of the optimization process,

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07.
https://doi.org/10.1145/3319619.3326802

we need to find a way to adapt the search algorithm to the current situation. To do this, we propose an online parameter tuning scheme that finds the sets of parameters that work best during the different stages of optimization. These kinds of methods have been studied in great detail such as in [7], [12], and [8], and some have been successfully implemented within optimization algorithms, such as in [10].

In our paper, the method we used to search for optimal sets of parameters was another "Parameter Search" Two Mode algorithm. Throughout the optimization process, 4 sets of parameters were simultaneously tested as possible candidates. All parameter sets share information among each other about their sample positions and the objective functions associated with them. Every 5 sequential steps, the performance of the parameter sets was evaluated, using the formula in Table 2 (line 28). The best ones were kept and the bad ones replaced with new suggestions, generated by the "Parameter Search" Two Mode algorithm. The parameters controlling this second level Two Mode algorithm were tuned by hand to work best on the space of parameters. Though we used a secondary Two Mode algorithm for the online parameter tuning, other methods can be used as well (such as batch Bayesian optimization, as described in [19]).

Regardless of the tuning method we use, we cannot avoid the possibility that some parameter sets will perform better than others during different stages of the optimization. Thus, it would be beneficial to find a way to allocate fewer resources to these "bad apples" and allocate more resources to better ones.

2.2 PARALLEL OPTIMIZATION WITH RESOURCE ALLOCATION

To allocate resources in a way that transfers more resources to better performing sets of parameters (and away from bad ones), we propose a resource allocation system called a "Conductor". This method consists of a linear set of rate equations, which form a performance-based resource allocation system. In this system, the better a set of parameters performs during a certain iteration, the more resources it will receive during the next.

Let us define $N_{tot} = \Sigma_{i=1}^{m} N^i$ as the total number of parallel samples in each iteration, across all m parameter sets. Let $\mathbf{N}$ be a vector representing the current allocation of parallel samples between m sets of parameters, such that $\Sigma_i N^i = N_{tot}$. The resource allocation of iteration t+1 is determined by:

$$\mathbf{N_{t+1}} = \mathbf{N_t} - KM^{-1}\mathbf{\Delta O_{Pbest_t}} - K_0 M^{-1}(\mathbf{N_t} - \mathbf{N_0}), \quad (1)$$

where K symbolizes the interaction matrix, of which all off-diagonal elements are equal to a constant k and all diagonal elements are equal to $(m-1)k$. K_0 is the self interaction, a diagonal matrix with positive elements k_0. M^{-1} is the inverse mass matrix, a diagonal matrix which represents how difficult it is to change the resources of a certain algorithm. $\mathbf{\Delta O_{Pbest_t}}$ represents a vector of the best change in the black box objective function we are trying to minimize, found by the different parameter sets, during iteration t. $\mathbf{N_0}$ is a vector that represents the initial allocation of samples among parameter sets.

This choice for constructing the matrices aims to conserve the total number of samples in each iteration, across all parameter sets. This means that $\Sigma_i N_t^i = \Sigma_i N_0^i = N_{tot}$ for any iteration t. The first term removes resources from parameter sets that perform poorly on minimization (positive ΔO_t) and towards sets that perform well (negative ΔO_t). The second term causes a gradual decay back to the initial resource allocation. This means that the system will gradually return to the initial allocation, unless some of the parameter sets consistently perform better (or worse) than the others. To prevent $\mathbf{\Delta O_{Pbest_t}}$ from becoming very big or very small, we found it beneficial to first normalize this vector using the L2 norm.

2.3 PSEUDO-CODE

We provide the pseudo-code of the Adaptive Two Mode Algorithm in Table 2. The notation used in the table, that has not been mentioned in the previous section, is specified below:

$\mathcal{R}_t$ = Amplitude of the isotropic search, at iteration t.
$\mathcal{R}_{max}$ = Maximal amplitude of the isotropic search.
$\mathcal{D}_t$ = Amplitude of the directional search, at iteration t.
$\mathbf{X_{1,..,8}}, O_{1,..,8}$ = Positions of the best eight samples and their associated objective values.
$\mathbf{X_{i_t}^P}$ = Parameter set i at iteration t.
$\mathbf{X_{1,2_t}^P}$ = Best two parameter sets at iteration t.
$\mathbf{O}^P$ = Vector of objective values for the parameter sets.
$\mathbf{Y}$ = Vector containing the adaptation parameters, used to adapt the search parameters.
r = Growth factor for the amplitude of the isotropic search.
d = Growth factor for the amplitude of the directional search.
G_r = Growth rate for the amplitude of the isotropic search.
T_r = Oscillation period for the amplitude of the isotropic search.
G_d = Growth rate for the amplitude of the directional search.
D_d = Decay rate for the amplitude of the directional search.
$\vec{A}$ = Search amplitude vector - controls the amplitude of the search for each parameter of the problem.
$\vec{S}$ = Exponentially weighted squared gradient.
$\Delta X_{min_i}^2$ = Minimal change to trigger growth of the directional mode.
fun = The function we are trying to minimize.
$crossover(\mathbf{X_{1,...,8_t}})$ = create new sample by randomly choosing values for each elements from $\mathbf{X_{1,...,8_t}}$, (uniformly distribution).
α = controls how much the search amplitude vector is allowed to deviate from 1.
β = the decay rate of the exponentially weighted average of $\vec{S}$.

Table 1: Parameter values and ranges

Parameter	Ranges for parameter	Value of adaptation parameter
G_r	$(10^{-2}, 10)$	5
T_r	$(10^{-1}, 10)$	5
$\mathcal{R}_{max}$	$(10^{-7}, 10)$	20
G_d	$(10^{-2}, 5)$	0.5
D_d	$(10^{-1}, 10^2)$	2
$\Delta X_{min_i}^2$	$(10^{-30}, 10^{-2})$	10^{-5}
α	$(10^{-5}, 10^{15})$	10^9
β	$(10^{-1}, 1)$	0.6

Table 2: Algorithm: Two Mode

1. Initialize random parameter sets $\mathbf{X^P_{1,\ldots,NumSets}}$ within ranges.
2. $\mathbf{N_0} = \mathbf{N_{tot}}$/Number of parameter sets.
3. $\mathbf{X_{1_0}}$, $\mathbf{X_{1_{-1}}}$, $\mathbf{X_{2,..,8_0}} = \mathbf{x0}$
4. $O1$, $O2 = fun(x0)$
5. **A, S = 0, 0**
6. r_i, $d_i = 0, 0$ for all i parameter sets.
7. **while** ending condition not met **do**
8. **for all** i parameter sets **do**
9. $\mathcal{R}_i = \mathcal{R}_{max_i} e^{(G_{r_i} \sin(mod(\frac{r_i \pi}{2\Gamma_{r_i}}, \frac{\pi}{2}) + 0.01) - 1)}$
10. $\mathcal{D}_i = \mathcal{R}_{max_i} e^{(G_{d_i} d_i - D_{d_i} r_i)}$
11. **for all** n samples in N_i **do**
12. **Mutation** $= \mathcal{R}_i$ ***random_vector*** $* \mathbf{A}_i$
13. $RandomSamples_i(n, :) = crossover(\mathbf{X_{1,\ldots,8_t}}) + \mathbf{Mutation} - \mathbf{X_{1_t}}$
14. $DirectionalSamples_i(n, :) = \mathcal{D}_i(\mathbf{X_{1_t}} - \mathbf{X_{1_{t-1}}}) \cdot randnum(0, 1)$
15. **end for**
16. $Samples_i = \mathbf{X_{1_t}} + DirectionalSamples_i + RandomSamples_i$
17. **end for**
18. $\mathbf{O} = fun(AllSamples)$
19. $\Delta O_{Pbest_i} =$ best change in O, from each parameter set i
20. $O_{Gbest} = min(\mathbf{O})$
21. Save new $\mathbf{X_{1,\ldots,8_t}}$
22. $\mathbf{S} = \beta \mathbf{S} + (1 - \beta) mean((\frac{\mathbf{O} - O_{Gbest}}{AllSamples_t - \mathbf{X_{1_t}}})^2)$
23. $\mathbf{A} = \frac{\alpha}{\sqrt{\mathbf{S} + \alpha^2}}$
24. $\mathbf{N} = \mathbf{N} - KM^{-1}\Delta\mathbf{O_{Pbest}} - K_0 M^{-1}(\mathbf{N} - \mathbf{N_0})$ (Update resource allocation)
25. **if** $(\mathbf{X_{1_t}} - \mathbf{X_{1_{t-1}}})^2 > \Delta X^2_{min_i}$ **do** $d_i += 1$, $r_i = 0$
26. **else do** $r_i += 1$, $d = 0$
27. **if** time to evaluate performance of parameter sets **do**
28. $\mathbf{O}^P = (mean(\Delta\mathbf{O_{Pbest}}) + min(\Delta\mathbf{O_{Pbest}}))/2$
29. Save best two parameter sets: $\mathbf{X^P_{1,2_t}}$
30. Repeat 9-16 (skip 11,15), using: $\mathbf{X^P_{1,2_t}}$, $\mathbf{X^P_{1_{t-1}}}$, and Y as parameters
31. **end if**
32. **end while**
33. **Output:** X_{1_t} (Best Solution found)

3 EXPERIMENTAL PROCEDURE AND CPU TIMING

In this experiment, we benchmarked the ATM algorithm on the BBOB noiseless test-bed. A fixed population of 64 samples (to be evaluated in parallel) was maintained throughout the optimization process. These samples were distributed among 4 sets of parameters, which were adapted in real-time using the secondary Two Mode algorithm, as discussed in Section 2.1. The resource allocation system described in Section 2.2 was used to divide samples among the different parameter sets.

In order to evaluate the CPU timing of the algorithm, we have run the ATM algorithm on the BBOB test suite [1] with restarts for a maximum budget equal to $2x10^5D$ function evaluations according to [2]. The python code was run on a Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz with 1 processor and 8 cores. The time per function evaluation for 2, 3, 5, 10, 20, and 40 dimensions, in units of $10^{-5} seconds$ equals: 4.1, 4.3, 4.5, 5.0, 6.0, and 7.8 , respectively.

4 RESULTS

Results of ATM from experiments according to [5] and [4] on the benchmark functions given in [1, 3] are presented in Figures 1, 2, 3, and 4 and in Tables 3 and 4. The experiments were performed and the plots were produced with COCO [6], version 2.1.

5 DISCUSSION

As can be seen from the results displayed in the Figure 1, the ATM performs much better on lower dimensional problems than higher dimensional ones. It manages to solve 23/24 problems in 2D, but only 8/24 in 40D, when given a budget of $2x10^5D$.

There are several possible reasons for this. First, because ATM attempts to find and exploit search directions that lead to improvement, it is challenged by problems that require many changes in the direction of search, such as f8, f9, and f12. A second reason, is that the lack of a co-variance matrix seems to limit the capabilities of the algorithm to adapt to situations in which the objective function changes very rapidly in some directions but slowly in others. Though this problem is partially dealt with by the use of the search amplitude vector $\vec{A}$ (as seen in Table 2), which allows the algorithm to solve all separable functions f1-f5, it does not always work well on rotated functions - such as f7 and f10.

As for multi-modal functions, ATM does manage to solve many of them in low dimensions ($D < 5$). Though it solves f3, f4 and f21 in all dimensions, it does not seem to scale well on other multimodal problems, that have narrow local minima such as f15 and f16. It is possible that a larger budget will allow the algorithm to make more progress on these problems. Additionally, performance could be improved by increasing the number of parameter sets used for online tuning and/or using a different online parameter tuning method.

6 CONCLUSIONS

The Adaptive Two Mode optimization algorithm performs relatively well on problems that do not have many narrow local minima and do not have non-separable directions in which the objective function changes much faster or slower than other directions. Because it does not require large matrix operations, the ATM has the potential to scale well to high dimensional problems with the previously stated properties. Though our results show that the internal runtime of the ATM scales slowly as a function of the dimensionality of the problems, this is not always true for the runtime to solve them, especially for problems with the properties stated above. We expect the ATM to be especially useful for optimizing high dimensional problems, in which the exact gradients are hard to calculate and/or do not provide useful information.

7 ACKNOWLEDGMENTS

The author would like to thank Brenda Rubenstein for all of her help and guidance developing this algorithm. The author would furthermore like to thank everyone who helped create the COCO framework [6]. It was amazingly straightforward and intuitive

to work with and was indispensable for the development of the algorithm.

8 SOURCE CODE

We made the source code available at:
https://github.com/BjBodner/ATM-optimization-algorithm
We hope this will enable users to test, apply and further develop the algorithm.

REFERENCES

[1] S. Finck, H. Hansen, R. Ros, and A. Auger. *Real-parameter black-box optimization benchmarking 2009: Presentation of the Noiseless Functions.* Technical Report 2009/20, Research Center PPE, 2009. .

[2] N. Hansen, A. Auger, O. Mersmann, T. Tusar, and D. Brockhoff. *COCO: A platform for comparing continuous optimizers in a black-box setting.* ArXiv e-prints, vol. arXiv:1603.08785, 2016.

[3] N. Hansen, S. Finck, R. Ros, and A. Auger. *Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions.* INRIA, Saclay, France, Tech. Rep. RR-6829, 2014.

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

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

[6] N. Hansen, A. Auger, O. Mersmann, T. TuÅąar, and D. Brockhoff. *2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting.* ArXiv e-prints arXiv:1603.08785 (2016).

[7] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stutzle. *ParamILS: An Automatic Algorithm Configuration Framework.* J. Artif. Intell. Res. (JAIR), vol. 36, pp. 267 - 306, 2009.

[8] M. Lopez-Ibanez, J. Dubois-Lacoste, T. Stutzle, and M. Birattari. *The irace Package: Iterated Racing for Automatic Algorithm Configuration.* J. RIDIA, Universite Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004, 2011.

[9] N. Hansen and S. Kern. *Evaluating the CMA Evolution Strategy on Multimodal Test Functions.* in PPSN, 2004, pp. 282-291.

[10] Ilya Loshchilov, Marc Schoenauer, Michele Sebag *BI-population CMA-ES Algorithms with Surrogate Models and Line Searches.* Genetic and Evolutionary Computation Conference (GECCO 2013), Jul 2013, Amsterdam, Netherlands. 2013.

[11] Sutskever, I., Martens, J., Dahl, G. , Hinton, G. *On the importance of initialization and momentum in deep learning.* In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML13, III - 1139 - III - 1147 (JMLR.org, 2013

[12] Dougal Maclaurin, David Duvenaud, Ryan P. Adams *Gradient-based Hyperparameter Optimization through Reversible Learning.* Proceedings of the 32 nd International Conference on Machine Learning, Lille, France, 2015.JMLR:WCP volume37.

[13] Dan Vladimir Nichita, Susana Gomez, Eduardo Luna *Multiphase equilibria calculation by direct minimization of Gibbs free energy with a global optimization method.* Computers and Chemical Engineering 26 (2002) 1703/1724

[14] Bin Qian, Angel R. Ortiz, David Baker *Improvement of comparative model accuracy by free-energy optimization along principal components of natural structural variation.* PNAS October 26, 2004, vol. 101, no. 43, 15347

[15] Shai Shalev-Shwartz, Ohad Shamir, Shaked Shammah *Failures of Gradient-Based Deep Learning.* ICML'17 Proceedings of the 34th International Conference on Machine Learning - Volume 70, Pages 3067-3075

[16] Songqing Shan , G. Gary Wang *Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions.* Struct Multidisc Optim (2010) 41:219-241 DOI 10.1007/s00158-009-0420-2

[17] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, Nando de Freitas *Bayesian Optimization in a Billion Dimensions via Random Embeddings.* Journal of Artificial Intelligence Research 55 (2016) 361-387

[18] C. G. Broyden *The convergence of a class of double-rank minimization algorithms.* Journal of the Institute of Mathematics and Its Applications, 6:76-90, 1970.

[19] Wu, Jian and Frazier, Peter *The Parallel Knowledge Gradient Method for Batch Bayesian Optimization.* Advances in Neural Information Processing Systems 29 (2016) pages 3126–3134, Curran Associates, Inc.

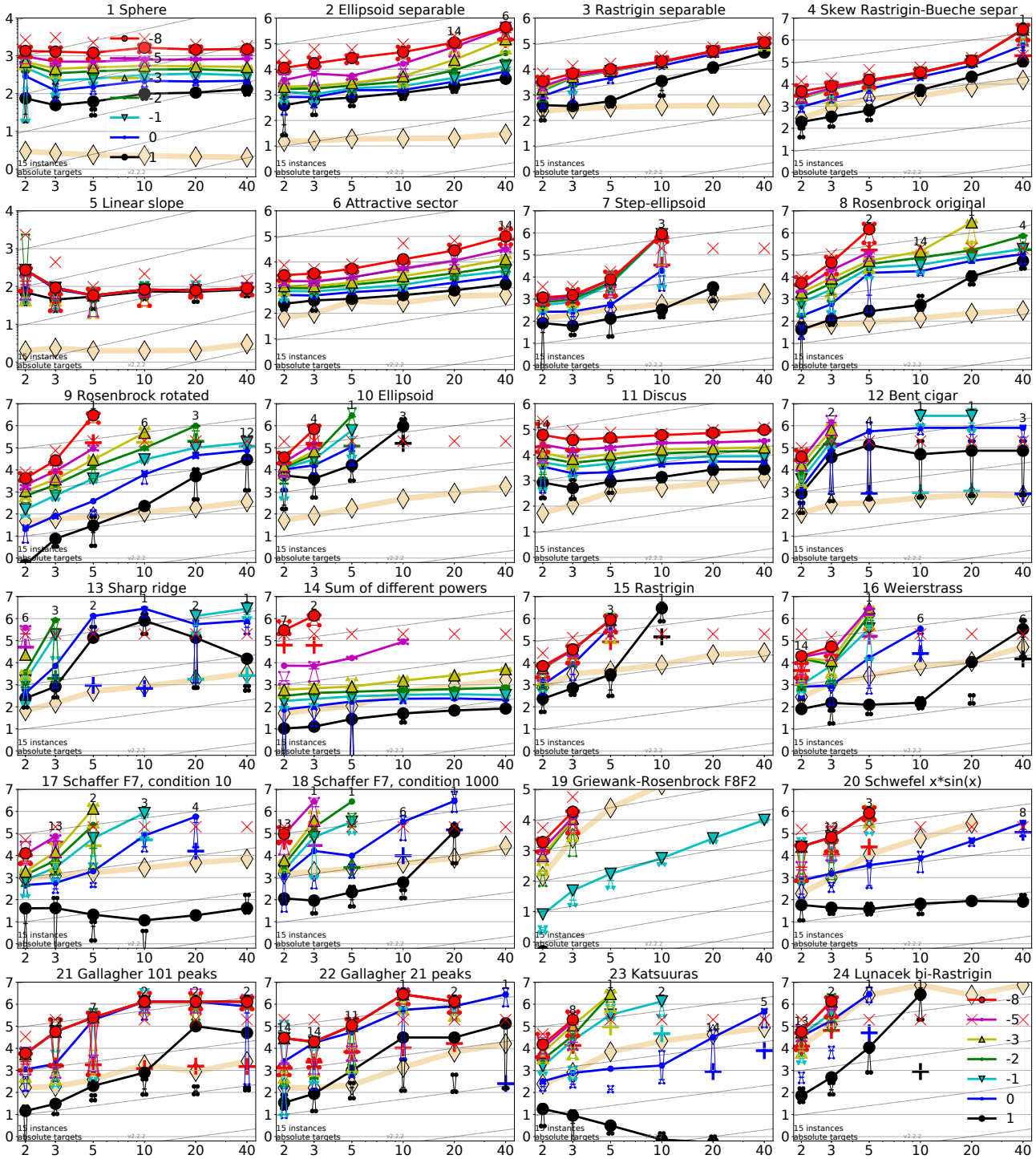


Figure 1: Scaling of runtime with dimension to reach certain target values Δf. Lines: average runtime (aRT); Cross (+): median runtime of successful runs to reach the most difficult target that was reached at least once (but not always); Cross (×): maximum number of f-evaluations in any trial. Notched boxes: interquartile range with median of simulated runs; All values are divided by dimension and plotted as $\log_{10}$ values versus dimension. Shown is the aRT for fixed values of $\Delta f = 10^k$ with k given in the legend. Numbers above aRT-symbols (if appearing) indicate the number of trials reaching the respective target. The light thick line with diamonds indicates the best algorithm from BBOB 2009 for the most difficult target. Horizontal lines mean linear scaling, slanted grid lines depict quadratic scaling.

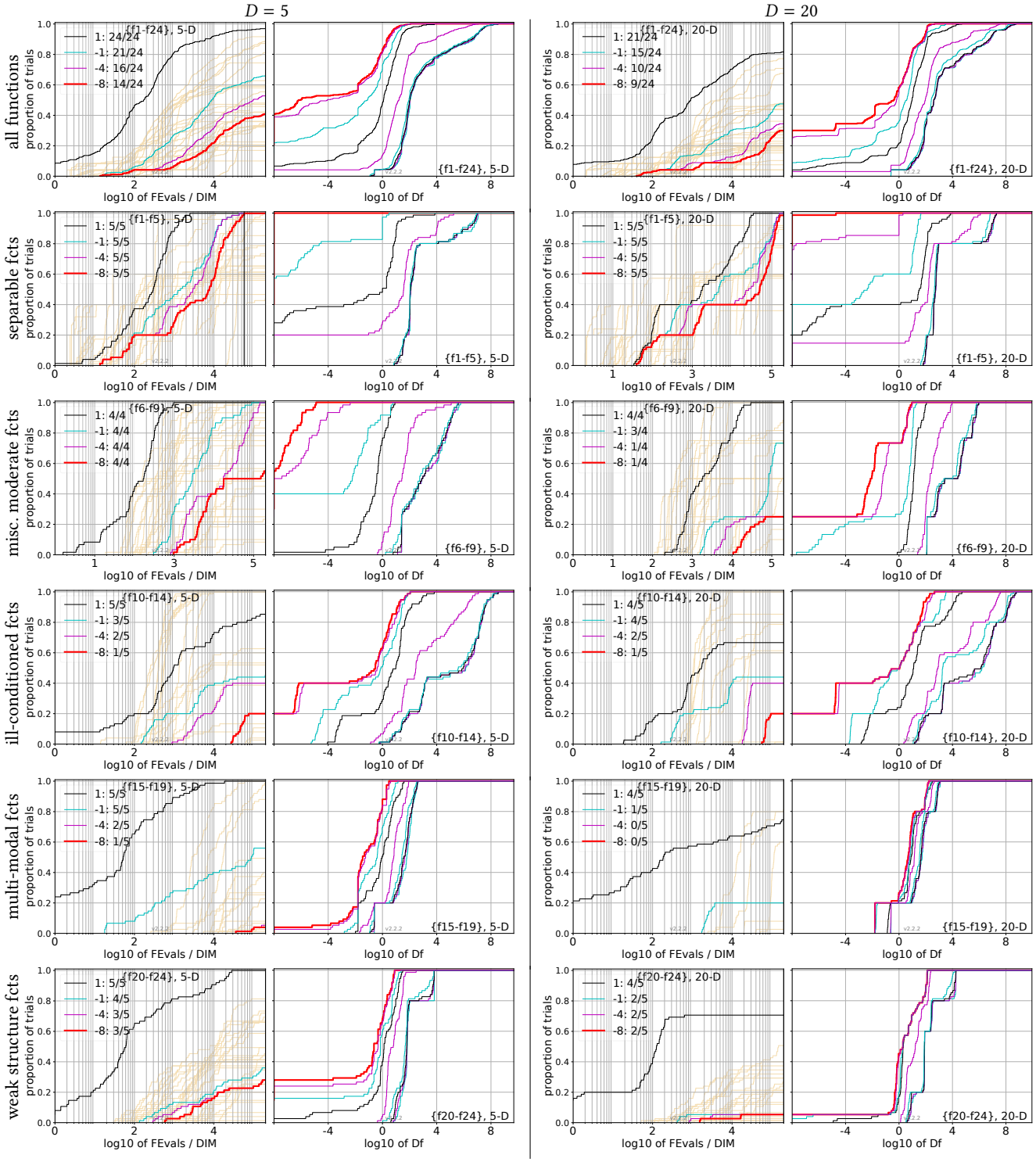


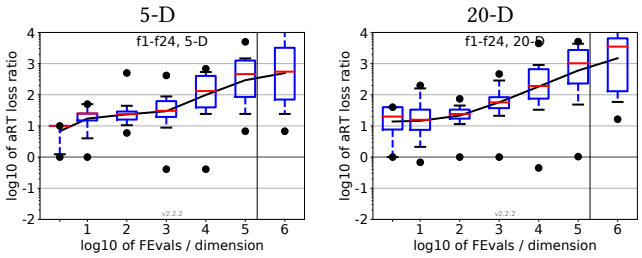
Figure 2: Empirical cumulative distribution functions (ECDF), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Left subplots: ECDF of the number of function evaluations (FEvals) divided by search space dimension D, to fall below $f_{\text{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. The thick red line represents the most difficult target value $f_{\text{opt}} + 10^{-8}$. Legends indicate for each target the number of functions that were solved in at least one trial within the displayed budget. Right subplots: ECDF of the best achieved Δf for running times of $0.5D, 1.2D, 3D, 10D, 100D, 1000D, \ldots$ function evaluations (from right to left cycling cyan-magenta-black…) and final Δf-value (red), where Δf and Df denote the difference to the optimal function value. Light brown lines in the background show ECDFs for the most difficult target of all algorithms benchmarked during BBOB-2009.

Δf	1e+1	1e+0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f_1	11	12	12	12	12	12	12	15/15
	66(69)	148(133)	259(109)	387(110)	490(164)	673(286)	919(202)	15/15
f_2	83	87	88	89	90	92	94	15/15
	1597(2070)	6214(6483)	17669(17126)	79700(73035)	78820(31945)	∞	∞ *500006*	0/15
f_3	716	1622	1637	1642	1646	1650	1654	15/15
	86(135)	4809(2774)	∞	∞	∞	∞	∞ *500022*	0/15
f_4	809	1633	1688	1758	1817	1886	1903	15/15
	76(98)	4766(7425)	∞	∞	∞	∞	∞ *500020*	0/15
f_5	10	10	10	10	10	10	10	15/15
	109(70)	134(73)	138(78)	140(81)	140(79)	140(63)	140(95)	15/15
f_6	114	214	281	404	580	1038	1332	15/15
	34(45)	28(28)	36(16)	38(11)	38(21)	42(24)	54(29)	15/15
f_7	24	324	1171	1451	1572	1572	1597	15/15
	85(71)	21(19)	29(41)	96(128)	157(227)	157(201)	155(231)	15/15
f_8	73	273	336	372	391	410	422	15/15
	41(43)	593(1135)	602(979)	712(1089)	901(758)	2471(1738)	4023(3764)	1/15
f_9	35	127	214	263	300	335	369	15/15
	20(26)	38(56)	206(158)	433(207)	923(757)	3211(3365)	7276(12103)	1/15
f_{10}	349	500	574	607	626	829	880	15/15
	1034(1297)	2408(4032)	6683(5444)	∞	∞	∞	∞ *500005*	0/15
f_{11}	143	202	763	977	1177	1467	1673	15/15
	40(28)	181(62)	111(48)	134(43)	232(90)	506(404)	∞ *500009*	0/15
f_{12}	108	268	371	413	461	1303	1494	15/15
	376(1639)	1007(1052)	4563(4097)	17156(18766)	∞	∞	∞ *500011*	0/15
f_{13}	132	195	250	319	1310	1752	2255	15/15
	280(799)	1074(1078)	2852(2360)	4392(3812)	5558(9256)	∞	∞ *500035*	0/15
f_{14}	10	41	58	90	139	251	476	15/15
	26(48)	58(36)	77(52)	77(67)	103(83)	736(627)	∞ *500006*	0/15
f_{15}	511	9310	19369	19743	20073	20769	21359	14/15
	177(146)	73(30)	82(53)	81(96)	80(74)	78(83)	76(93)	3/15
f_{16}	120	612	2662	10163	10449	11644	12095	15/15
	30(78)	93(81)	111(126)	97(103)	345(347)	∞	∞ *500018*	0/15
f_{17}	5.0	215	899	2861	3669	6351	7934	15/15
	159(20)	107(193)	130(95)	157(141)	356(555)	∞	∞ *500021*	0/15
f_{18}	103	378	3968	8451	9280	10905	12469	15/15
	73(26)	165(215)	151(215)	904(1376)	∞	∞	∞ *500023*	0/15
f_{19}	1	1	242	1.0e5	1.2e5	1.2e5	1.2e5	15/15
	2.0	2.0	154(10)	∞	∞	∞	∞ *500010*	0/15
f_{20}	16	851	38111	51362	54470	54861	55313	14/15
	29(29)	80(82)	94(142)	70(83)	66(60)	66(27)	65(41)	2/15
f_{21}	41	1157	1674	1692	1705	1729	1757	14/15
	42(10)	206(290)	260(254)	258(269)	257(221)	254(247)	251(194)	14/15
f_{22}	71	386	938	980	1008	1040	1068	14/15
	110(175)	115(109)	152(170)	149(319)	147(197)	154(297)	160(290)	15/15
f_{23}	3.0	518	14249	27890	31654	33030	34256	15/15
	12(26)	10(11)	28(20)	146(194)	245(229)	241(288)	∞ *500038*	0/15
f_{24}	1622	2.2e5	6.4e6	9.6e6	9.6e6	1.3e7	1.3e7	3/15
	109(138)	∞	∞	∞	∞	∞	∞ *500023*	0/15

Table 3: Average running time (aRT in number of function evaluations) divided by the aRT of the best algorithm from BBOB 2009 in dimension 5. The aRT and in braces, as dispersion measure, the half difference between 90 and 10%-tile of bootstrapped run lengths appear in the second row of each cell, the best aRT (preceded by the target Δf-value in *italics*) in the first. #succ is the number of trials that reached the target value of the last column. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Bold entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm from BBOB 2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the ↓ symbol, with Bonferroni correction by the number of functions (24).
Data produced with COCO v2.2.1.10

Δf	1e+1	1e+0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f_1	43	43	43	43	43	43	43	15/15
	103(41)	236(150)	361(160)	470(254)	566(251)	789(442)	1123(288)	15/15
f_2	385	386	387	388	390	391	393	15/15
	∞	∞	∞	∞	∞	∞	∞ *2000006*	0/15
f_3	5066	7626	7635	7637	7643	7646	7651	15/15
	∞	∞	∞	∞	∞	∞	∞ *2000017*	0/15
f_4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
	∞	∞	∞	∞	∞	∞	∞ *2000015*	0/15
f_5	41	41	41	41	41	41	41	15/15
	110(65)	124(46)	130(67)	130(79)	130(85)	130(52)	130(70)	15/15
f_6	1296	2343	3413	4255	5220	6728	8409	15/15
	69(54)	121(91)	192(238)	290(292)	510(613)	2196(2014)	∞ *2000011*	0/15
f_7	1351	4274	9503	16523	16524	16524	16969	15/15
	1262(988)	∞	∞	∞	∞	∞	∞ *2000089*	0/15
f_8	2039	3871	4040	4148	4219	4371	4484	15/15
	255(230)	571(410)	1101(977)	∞	∞	∞	∞ *2000007*	0/15
f_9	1716	3102	3277	3379	3455	3594	3727	15/15
	86(136)	477(491)	2888(1687)	∞	∞	∞	∞ *2000008*	0/15
f_{10}	7413	8661	10735	13641	14920	17073	17476	15/15
	∞	∞	∞	∞	∞	∞	∞ *2000006*	0/15
f_{11}	1002	2228	6278	8586	9762	12285	14831	15/15
	106(56)	113(65)	127(171)	435(280)	1074(1483)	∞	∞ *2000011*	0/15
f_{12}	1042	1938	2740	3156	4140	12407	13827	15/15
	214(203)	307(518)	670(585)	5061(6513)	∞	∞	∞ *2000029*	0/15
f_{13}	652	2021	2751	3507	18749	24455	30201	15/15
	254(39)	432(558)	582(373)	1196(837)	738(1067)	∞	∞ *2000031*	0/15
f_{14}	75	239	304	451	932	1648	15661	15/15
	36(37)	42(20)	63(12)	94(31)	286(217)	∞	∞ *2000012*	0/15
f_{15}	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
	∞	∞	∞	∞	∞	∞	∞ *2000014*	0/15
f_{16}	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
	230(308)	∞	∞	∞	∞	∞	∞ *2000057*	0/15
f_{17}	63	1030	4005	12242	30677	56288	80472	15/15
	83(41)	∞	∞	∞	∞	∞	∞ *2000008*	0/15
f_{18}	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
	16263(18562)	∞	∞	∞	∞	∞	∞ *2000010*	0/15
f_{19}	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
	2.0	2.0	1.0(3)	∞	∞	∞	∞ *2000005*	0/15
f_{20}	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
	29(15)	610(878)	∞	∞	∞	∞	∞ *2000022*	0/15
f_{21}	561	6541	14103	14318	14643	15567	17589	15/15
	122(300)	235(248)	139(114)	137(176)	135(111)	127(112)	113(107)	12/15
f_{22}	467	5580	23491	24163	24948	26847	1.3e5	12/15
	158(221)	211(216)	145(116)	142(113)	138(63)	133(49)	36(27)	5/15
f_{23}	3.0	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
	16(16)	29(20)	∞	∞	∞	∞	∞ *2000124*	0/15
f_{24}	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
	∞	∞	∞	∞	∞	∞	∞ *2000016*	0/15

Table 4: Average running time (aRT in number of function evaluations) divided by the aRT of the best algorithm from BBOB 2009 in dimension 20. The aRT and in braces, as dispersion measure, the half difference between 90 and 10%-tile of bootstrapped run lengths appear in the second row of each cell, the best aRT (preceded by the target Δf-value in *italics*) in the first. #succ is the number of trials that reached the target value of the last column. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Bold entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm from BBOB 2009, with $p = 0.05$ or $p = 10^{-k}$ when the number $k > 1$ is following the ↓ symbol, with Bonferroni correction by the number of functions (24).
Data produced with COCO v2.2.1.10



f_1–f_{24} in 5-D, maxFE/D=183935						
#FEs/D	best	10%	25%	**med**	75%	90%
2	2.0	2.4	10	10	10	10
10	2.0	3.3	17	25	50	50
100	14	19	31	50	98	1.8e2
1e3	20	30	51	87	1.4e2	1.8e2
1e4	13	46	1.1e2	1.7e2	3.7e2	6.4e2
1e5	35	83	1.4e2	2.8e2	1.2e3	1.9e3
1e6	65	1.0e2	1.7e2	8.3e2	1.0e4	1.4e4
RL_{US}/D	1e5	1e5	1e5	1e5	1e5	1e5

f_1–f_{24} in 20-D, maxFE/D=164187						
#FEs/D	best	10%	25%	**med**	75%	90%
2	2.0	2.0	13	40	40	40
10	0.80	2.0	14	37	2.0e2	2.0e2
100	2.0	9.3	28	53	89	4.6e2
1e3	2.0	26	56	99	2.6e2	5.1e2
1e4	2.0	1.2e2	1.8e2	3.0e2	6.7e2	1.5e3
1e5	5.8	1.3e2	3.9e2	7.6e2	2.3e3	5.4e3
1e6	47	1.1e2	1.3e3	5.5e3	1.4e4	3.5e4
RL_{US}/D	1e5	1e5	1e5	1e5	1e5	1e5

Figure 3: aRT loss ratio versus the budget in number of f-evaluations divided by dimension. For each given budget FEvals, the target value f_t is computed as the best target f-value reached within the budget by the given algorithm. Shown is then the aRT to reach f_t for the given algorithm or the budget, if the best algorithm from BBOB 2009 reached a better target within the budget, divided by the aRT of the best algorithm from BBOB 2009 to reach f_t. Line: geometric mean. Box-Whisker error bar: 25-75%-ile with median (box), 10-90%-ile (caps), and minimum and maximum aRT loss ratio (points). The vertical line gives the maximal number of function evaluations in a single trial in this function subset. See also Figure 4 for results on each function subgroup.
Data produced with COCO v2.2.1.10

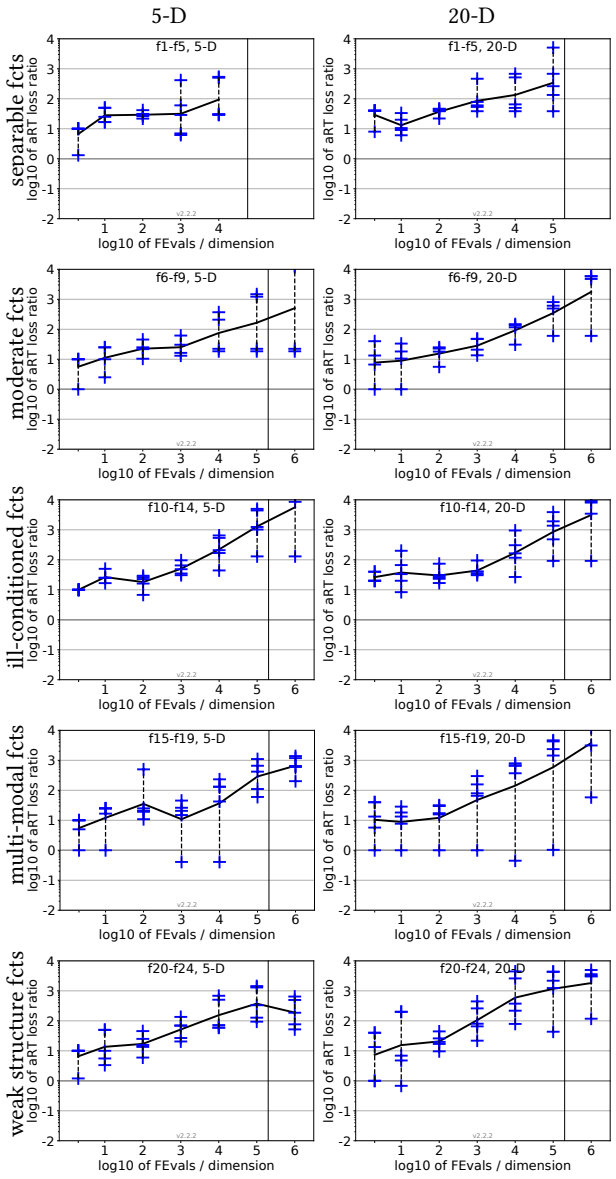


Figure 4: aRT loss ratios (see Figure 3 for details). Each cross (+) represents a single function, the line is the geometric mean.