Black-Box Optimization Benchmarking for $(\mu/\mu, \lambda) - \sigma$ SA-ES on the Noiseless Testbed

Sihem Abdoun, Stephen Batifol, Nicolas Bougie, Nawel Medjkoune and Abdelhadi Temmar

ABSTRACT

In this paper we present an implementation of a non benchmarked evolutionary algorithm $(\mu/\mu\overset{+}{,}\lambda) - \sigma SA - ES$ to solve a black-box optimization problem in a continuous domain. We benchmarked the algorithm using the COCO platform which provides the necessary tools for comparing continuous optimizer in a black-box scenario. We draw experiments through many parameters and compare our algorithm with other algorithms implemented for the same goal.

Keywords

Black-box optimization, Evolutionary Algorithms, COCO platform, Benchmarking, Continuous Optimization

1. INTRODUCTION

1.1 Project Context

In a black-box optimization problem, the function to minimize is unknown. This function is non-linear, non separable and can be non-convex, multimodal or ill-conditioned. In this context, a naive random search or a deterministic search takes too long. In many case, an evolutionary algorithm can be used.

From the algorithm pointview, evolution strategies are optimization methods that sample new candidates solutions stochastically. Evolution strategies are based on Darwinian principles : under environmental pressure, the population of individuals mute to adapt to their environment and thus survive.

We have implemented the evolutionary algorithm described in [3], we chose to implement the given algorithm with Python. The algorithm had his parameter control via self-adaptation. We then ran a suite of test functions in order to benchmark the performance of the implemented algorithm.

As central performance measure of algorithms COCO compute the "rutime" measured by the number of evaluations performed on the problem. In other words "runtime" is defined by the number of calls to the objective function until a target value is hit. This performance measure is independent from the used programming language and the material running the algorithm.

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

1.2 Project Goal

In order to know which algorithm to use for a particular numerical black-box optimization problem, we have to benchmark multiple algorithms. Thus we can compare the performances on different problems with different properties and shapes. For this particular project, we have implemented the $(\mu/\mu, \lambda)$–σSA–ES [3] , a non benchmarked algorithm and we benchmarked it with COCO [5]. We then compare this algorithm with two baseline algorithms, BIPOP-CMA-ES [2] and BFGS [8], along with another implementation of the same algorithm.

We organized this report into several sections, each section addresses an important point of this work. First, in section 2 we describe the implemented algorithm and give the pseudo-code as it was done in the original paper. In section 3, implementation and configuration details of the algorithm are explained. Section 4 summarizes details about the timing experiments. In section 5 we sum up the main results we got through several experiments. Finally, results are discussed in section 6 and suggestions about further investigation are given.

2. ALGORITHM DESCRIPTION

The algorithm studied in this project is $(\mu/\mu, \lambda) - \sigma SA - ES$ [3]. The name indicates that the parent population contains μ individuals, for the recombination μ are used and λ offsprings are generated at each iteration. The "+" indicates that the age is not used for the selection and $-\sigma SA$ that the σ is generated via self adaptation. We will discuss later about the properties of theses variables. Prior to implement the algorithm, we first had to understand it. We proceeded by understanding what an evolutionary algorithm consist of and the mechanism behind the template 2 of the paper which corresponds to the pseudo-code of the evolution strategy that is used in the article.

These algorithms are most commonly population-based where each individual of the population is considered as a candidate solution to the problem and its quality is measured by a loss function (objective function). The concept of the algorithm is close to biological evolution and Darwin's theory. The initial population is randomly generated (origin of life). This population has (probably) a very low evaluation score and is weak (relative to survival). The offsprings (second generation) are generated from one or several parents. These descendants inherit genes from their parents and then mutate their genes . Only the strongest individuals survive (the ones with the best evaluation score). This leads to a stronger and stronger population, in other words : get closer to the optimal solution. The purpose is then to improve the solutions iteratively by generating new solutions through stochastic mechanisms such as: selection, recombination or mutation.

The pattern of evolutionary algorithm is:

Algorithm 1: $(\mu/\mu, \lambda) - ES$

Input: n,$\lambda \in \mathbb{N}^+$
1 $x \in \mathbb{R}^+$
2 $P \leftarrow \{\}$
3 s;
4 **while** *not happy* **do**
5 $\quad i \leftarrow i + 1$
6 $\quad$ **for** $k \leftarrow 1$ ***to*** λ **do**
7 $\quad\quad s_k \leftarrow mutate_s(s)$
8 $\quad\quad x_k \leftarrow mutate_x(s_k, x)$
9 $\quad\quad P \leftarrow P \cup (x_k, s_k, f(x_k))$
10 $\quad P \leftarrow select\_by\_age(P)$
11 $\quad (x, s) \leftarrow recombine(P, x, s)$

Figure 1 shows the template 2 exposed in the paper. The particularity of this template is the use of a single parental centroid (x, s), which simplify the algorithm. We shall use this single parental centroid to create a new generation of offspring. "S_k", more commonly called "standard deviation", is used to generate a new offspring X_k. Intuitively, The bigger S_k is, the bigger the distance between X and X_k is. This is why σ and σ_k change through iteration, we want this parameter to adapt over time, this process is called Self Adaptation. We use X and S to generate the set $(x_k, s_k, f(x_k))$ for k $=\{0...\lambda\}$. At line 9 and 10 of figure, the population is updated using the new samples. At line 11, X and S, the parental centroid, is updated. At line 11, X and S, the parental centroid, is updated.

Algorithm 2: $(\mu/\mu, \lambda) - \sigma SA - ES$

Input: $n \in \mathbb{N}^+, \lambda >= 5n, \mu \approx \lambda/4, \tau \approx 1/\sqrt{n}, \tau_i \approx 1/n^{1/4}$
1 $x \in \mathbb{R}^n$
2 $\sigma \in \mathbb{R}^n_+$
3 **while** *not happy* **do**
4 $\quad$ **for** $k \leftarrow 1$ ***to*** λ **do**
5 $\quad\quad \xi_k \leftarrow \tau\mathcal{N}(0, 1)$
6 $\quad\quad \xi_{\mathbf{k}} \leftarrow \tau_i\mathcal{N}(0, \mathbf{I})$
7 $\quad\quad z_k \leftarrow \mathcal{N}(0, \mathbf{I})$
8 $\quad\quad \boldsymbol{\sigma_k} \leftarrow \sigma \circ exp(\boldsymbol{\xi_k}) \times exp(\xi_k)$
9 $\quad\quad x_k \leftarrow x + \boldsymbol{\sigma_k} \circ \boldsymbol{z_k}$
10 $\quad P \leftarrow select\_\mu\_best(\{(\boldsymbol{x}_k, \boldsymbol{\sigma}_k, f(\boldsymbol{x}_k)) | 1 \le k \le \lambda\})$
11 $\quad \boldsymbol{\sigma} \leftarrow \frac{1}{\mu}\sum_{\boldsymbol{\sigma}_k \in \mathbb{P}} \boldsymbol{\sigma}_k$
12 $\quad \boldsymbol{x} \leftarrow \frac{1}{\mu}\sum_{\boldsymbol{x}_k \in \mathbb{P}} \boldsymbol{x}_k$

Figure 2 shows the pseudocode of the $(\mu/\mu \overset{+}{,} \lambda) - \sigma SA - ES$ algorithm. The goal of the algorithm is to minimize a black box problem. More precisely, the algorithm tries to find a X (entry of f) which will minimize f(X). For this reason, we first have to initialize randomly X and try to find near this parental centroid a better solution. The algorithm is very close to 1.

The new idea of the self-adaptation model is to adapt and mutate the step-size in a similar way to x. This improvement is crucial to solve ill-conditioned problems and get a faster exploration (of the search space). Those classes of problems have large ratio between longest and shortest axis of ellipsoid (level sets). A conventional algorithm fails because it explores too much / not enough dimensions of space. This is even more true for ill-conditioned problems. The function to optimize is unknown so there is no way to predict the optimal step-size. The best solution is achieved with an adaptive algorithm. Based on the previous solution, the model adapt the step-size for each component (undergo a common mutation) to optimally explore the search space. In other words, for the large eigenvalue of the hessian, the step size will be large (to quickly move toward the solution) and for a small eigenvalue the step size will be small. The hessian matrix is not available (function is unkwown and too complex). The algorithm approximates the "level sets" of the function and thus adapt the step-sizes at each iteration.

To understand deeper this algorithm, we need to understand how the mutation, selection and recombination are made.

First of all, the mutation. During this step, we create a new generation (children) by applying random changes following a normal distribution. The normal distribution is often observed as phenotypic traits (in nature), it's stable distribution with finite variance and the maximum entropy distribution with finite variance. We can see at line 9, that we apply 2 mutations to σ (step size). One mutation which is common to all components of the vector (the scalar computed at line 6), and one which is different for each component of (the vector computed at line 8). For x, we apply a different mutation for each component (line 10).

The mutation is controlled by a multivariate normal distribution (line 8,9,10). The first step (line 8) is computing a normal distribution for each individual. The covariance matrix of the distribution determines the shape of the ellipsoid. In our case, the covariance matrix is the identity matrix which mean that no direction are advantaged. Then, after updating the σ_k, the algorithm explores the spaces around the centroid (best actual solution), by adding the normal distribution to the centroid. We perform this operation because the normal distribution is centered (mean) in 0. To determine the shape of the level sets of the normal distribution, we multiply $\mathcal{N}(0, 1)$ by σ_k. Each dimension is affected by the σ_k. For an ill conditioned problem, σ_k stretched the ellipsoid for large eigenvalue of the hessian matrix (speed up the speed of exploration in this direction), for a small eighenvalue, the ellipsoid is tightened in the direction. This way of calculating the normal distribution is a major issue for functions with correlated components. Use of a diagonal matrix with correlated component is not efficient for space exploration.

The sign of $\boldsymbol{\sigma_k} \circ \boldsymbol{z_k}$ indicates for each component of X which direction to follow.

Once the mutation is done, we shall do the selection. Individual's selection is done by taking the μ best individuals among the lambda individuals of the new generation. To select the best individuals, the algorithm calls the objective function, in other words the "complexity" is λ per loop. Therefore, the maximum number of iterations is (budget/

λ). The selection of the σ_k is implicitly performed at this step. In this algorithm we let individuals die after one iteration.

The recombination is simple. We compute the average of the new population to update the parental centroid (line 13). A similar technique is used to update σ (the step size,line 12). However, the application of mutation and recombination introduces a moderate bias such that λ tends to increase. In order to achieve stable behavior of λ, the number of parents μ must be large enough, which is reflected in the setting of λ.

In brief, the algorithm mutates the actual best solution (centroid) to explore the space around this solution. The exploration probability is not symmetric and depends of the step size (σ). Then, it selects the μ best solutions and recombine them in a single vector (the centroid) and σ_k in a single vector. By performing this steps, we hope the solution (centroid) get closer to the optimal solution.

3. ALGORITHM IMPLEMENTATION AND CONFIGURATION DETAILS

The algorithm implementation was relatively simple. We had to follow instructions sequence of the algorithm $(\mu/\mu \overset{+}{,} \lambda) - \sigma SA - ES$ described in the section above, taking care to choose the right data structures. However, several questions were raised in order to get a good starting configuration. First of all, initialization of X and σ was done according to the *bbob* dataset documentation. We decided to initialize$X$ between -5 and 5 because it's shown that the optimum is always in this range. To start with a small value, σ is initialized between 0 and 0.01. As said before, we know the range of the optimum. Thus, we need a small standard deviation so we won't get far away from the optimum. The initial λ value is currently set to $5 \times n$, where n is the size of the vector X, according to the minimum value advised in [3] to avoid σ fluctuations . By the same way, the μ value is set to $\lambda/\ 4$, the τ value is set to $1/\ \sqrt{n}$ and the τ_i value is set to $1/\ n^{1/\ 4}$.

In our implementation we experiment three termination criteria inspired from [1]. The first criteria consists of checking if no budget is left, this criteria allows us to see how far the algorithm can go exploring the search space. The second criteria consists of checking if the standard deviation is 10^{-12} times smaller than the first deviation. The last criteria consists of checking if the range of the best objective function values of the last $10 + \lceil 30n/\ \lambda \rceil$ generation is zero, or the range of these function values and all function values of the recent generation is below 10^{-12}.

We implemented independent restarts for the algorithm to explore a larger number of function evaluations and draw experiments over λ . The algorithm restarts if there is still some budget left and one of the two others termination criteria is triggered. Thus, at each restart we increase the population size by replacing the initial λ by a value that is proportional to the dimension and number of restarts namely $5 \times n \times restarts$. Where $restart$ is the number of restarts carried out.

Regarding the budget, we tried to test our algorithm with multiple budgets values to see for which value(s) the algorithm converges. We also variate on our experiments the dimension's size of the problem. Details of these experimentation are given in Section 5 and discussed in Section 6.

4. CPU TIMING

In order to evaluate the CPU timing of the algorithm, we have run the $(\mu/\mu \overset{+}{,} \lambda) - \sigma SA - ES$ algorithm on the bbob test suite [6] with restarts. The maximum function evaluation are set to 5×10^4 and the experiments are done for $D = 2, 3, 5, 10, 20$.. The code in python was run on an Intel(R) Core(TM) i5-750 CPU @ 3.22GHz with 1 processor and 4 cores on Arch Linux. The initial solution is generated randomly in $[-5, 5]^D$ and a very small value for σ is generated randomly as well. The parameter λ is initially set to $5 * dim$. Finally, the maximum number of runs (restarts) in set to its default value: $1e9$.

5. RESULTS

Results from benchmarking $(\mu/\mu \overset{+}{,} \lambda) - \sigma SA - ES$ and comparing it to BIPOP-CMA-ES [2] and BFGS[8] on the benchmark functions given in [9] are presented in Figures 1, 2 and 3 and in Tables 1 and 2.

The experiments were performed with COCO [5], version 1.2.1.

The **average runtime (aRT)**, used in the figures and tables, depends on a given target function value, $f_t = f_{opt} + \Delta f$, 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[4, 7]. **Statistical significance** is tested with the rank-sum test for a given target $\Delta f_t = 10^{-8}$ as in Figure 1) using, for each trial, either the number of needed function evaluations to reach Δf_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. In the following, we will refer to our algorithm as ES and to the other team implementation as TANIT.

Among the 24 functions, 7 had a 100% success rate in 2-D, 5 in 5-D and 3 in 20-D. Two functions f3 and f4 seem to become practically unsolvable with increasing dimension, even BIPOP-C cannot solve these problems.

The scaling of the running time is between quadratic and cubic as we can see on Figure 1 in some cases worse but never better.

Figure 2 and Figure 3 show the empirical cumulative distribution functions (ECDFs) for 51 targets with precision between 100 and 10^{-8} in dimension 5 and 20 for the four algorithms. We can observe that in 5D, ES and TANIT are quite similar in the proportion of problems solved, which is expected because they both implement the same algorithm (with maybe different configuration and stopping criteria). For separable and moderate functions, ES solves up to 75% of the problems while BIPOP-C solves all problems in less time. The performance could be better with a bigger budget (10^6). For ill-conditioned and multi-modal functions, only 30% of the problems were solved, while BIPOP-C still reaches a 100% of targets. BFGS in this case solves 80% of ill-conditioned functions but only 18% of multi-modal functions. For the weakly structured multi-modal functions, no algorithm solves all problems. BIPOP-C still has the best performance with 90% while ES solves 30% with the 5.10^4 xD fixed budget. On the overall functions, ES and TANIT solve about half of the proportion of the algorithms while BIPOP-C and BFGS perform better with 90% and 55% respectively. We observe that BFGS is usually faster than ES in the first stages of the run but becomes stationary after that. ES catches in the last stages. This may be due to the restarts where the population size is doubled each time. Regarding dimension 20D, the differences we can observe is that performance of ES drops considerably for moderate functions (which is due to an anomaly since TANIT solves 60%) and multi-modal functions, and it stays low for other groups of functions. This can be observed in the overall functions plot, where proportion of problems solved is 25% for ES. Other algortihms also perform worse than 5D for separable functions and weakly structured functions.

6. DISCUSSION

We compared the $(\mu/\mu, \lambda) - \sigma SA - ES$ algorithm with two baseline algorithms (BIPOP-C and BFGS) and another implementation of the same algorithm (TANIT).

As expected, TANIT and ES perform relatively the same way on overall functions. Few significant differences were observed in 20D that can be due to different initialization and stopping criteria. Empirical results show that $(\mu/\mu, \lambda) - \sigma SA - ES$ is able to solve 7 functions out of 24 in 2-D, 5 in 5-D and 3 in 20-D. The system performs well on separable problems but poorly on ill-conditioned and multi-modal problems.

In comparison, the implementation of $(\mu/\mu, \lambda) - \sigma SA - ES$ by TANIT solves 7 functions in 2-D, 5 in 5-D and 4 in 20-D, the BIPOP-CMA-ES solves 23 functions in 2-D, 21 in 5-D and 18 in 20-D and finally, the BFGS solves 11 functions in 2-D, 7 in 5-D and 5 in 20-D. In order to solve more problems, we would need to increase the budget consequently, a budget of 5×10^5 would benefit the algorithm. Two points can be discussed regarding the results: First of all, $(\mu/\mu, \lambda) - \sigma SA - ES$ performs better on separable functions than on other functions, and this in both dimensions. This can be explained by the Axis-parallel mutation operator. In this case, variations of the variables are independent as the covariance matrix is a diagonal matrix and therefore variables are not correlated. With not separable functions, this algorithm performs poorly as the variations are made in different directions. This suggestion is further verified as we observe in the performance of BIPOP-CMA-ES. In this latter, the perturbation is drawn from multivariate normal distribution with a covariance matrix. As explained in section 2, this type of self adaptation mechanisms addresses the case where variables are correlated. We observe that it performs very well for moderate functions, ill-conditioned and multi-modal. Their performance for separable functions drops a little bit and is equivalent to $(\mu/\mu, \lambda) - \sigma SA - ES$. Second point, we observed that $(\mu/\mu, \lambda) - \sigma SA - ES$ was relatively slow than other algorithms. One possible explanation would be related to the evolution of σ. In the implemented self adaptation mechanism, step sizes are associated to individuals. In a new population, σ is generated by computing a mean over all σ_n. However, One value of σ which is good for mutating one individual might be not that good when applied to all the population. Therefore, calculating the mean of the standard deviation might not be a good idea. Instead, choosing σ from the best individual of the population can improve the progress of the algorithm.

7. CONCLUSION

The goal of this project was to implement $(\mu/\mu \overset{+}{,} \lambda) - SA - ES$ algorithm, presented on the paper, to benchmark it on the COCO platform and to study and compare the results with three other algorithms. The algorithm presented in this paper performed satisfactorily on many functions but gave poor results on others, As said before, we managed to solve only 7 out of 24 functions to a 100% (for the 2D dimension).

Questions remains about the results. For start, why do we have so poor results on the 20 dimension, despite the fact than the other group, working on the same algorithm, have better results on this same dimension. In contrary, our implementation produce better results for smaller dimensions. This can be explained by the fact that we have chosen different parameters, or different stopping criteria.

Another anomaly that we noticed is regarding the function 19 which always start in the first iterations of the run, with more than 0.2 % of instance solved. We do not managed to find out why we have this effect. We checked our initialization several times to see if we do it well randomly, and it's the case.

As mentioned in the paper, and as we can observe on the results, this algorithm is not perfect. It needs too much functions evaluation to perform correctly. The main issue must be the self-adaptation of σ. The authors talk about selection noise which "refers to the possibility that very good offspring may be generated with poor strategy parameter settings and vice versa". Algorithm 3 of the paper addresses this issue.

We have also thought about other improvements. $(\mu/\mu \overset{+}{,} \lambda) - SA - ES$ does not take age of individuals into account and chooses μ best of the λ individuals. Instead of that, it can be better to keep the best of the $\mu + \lambda$ individuals. Therefore, we will be sure to return the best result found during the process and not just during the last iteration. Also, implementing and testing different stopping criteria could significantly improve the results.

To summarize, the algorithm presented is far from being efficient, it needs too much resources to solve completely a function but by increasing the budget we can manage to perform correctly. Also, it would be better draw the perturbations from a normal distribution with covariance matrix. We could perform well on non-separable functions.

8. REFERENCES

[1] A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *2005 IEEE congress on evolutionary computation*, volume 2, pages 1769–1776. IEEE, 2005.

[2] N. Hansen. Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2389–2396. ACM, 2009.

[3] N. Hansen, D. V. Arnold, and A. Auger. Evolution strategies. In *Springer Handbook of Computational Intelligence*, pages 871–898. 2015.

[4] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2012: Experimental setup. Technical report, INRIA, 2012.

[5] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing

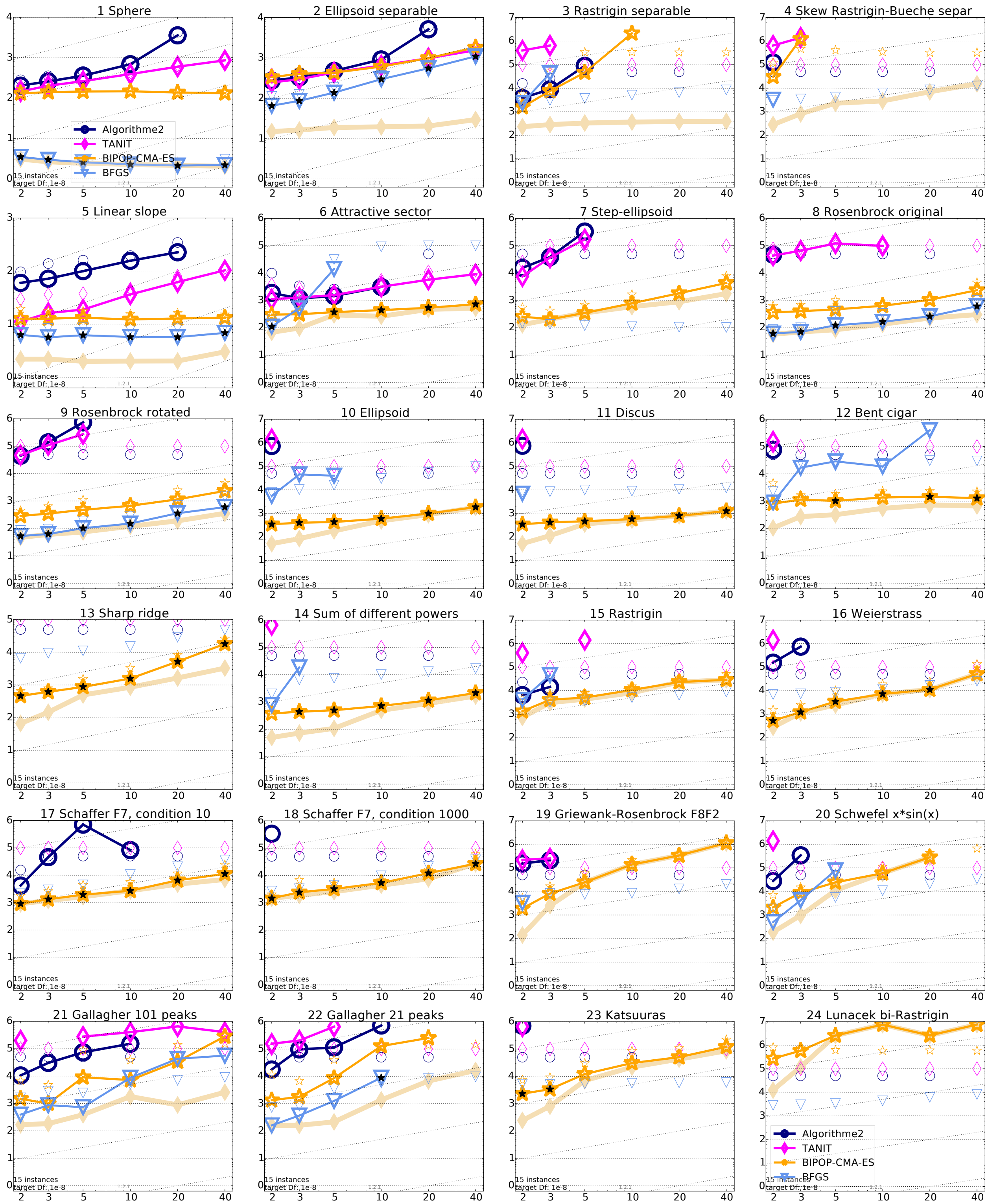

Figure 1: Average running time (aRT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of function 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: $\circ$: Algorithme2, $\diamond$: TANIT, $\star$: BIPOP-CMA-ES, $\triangledown$: BFGS

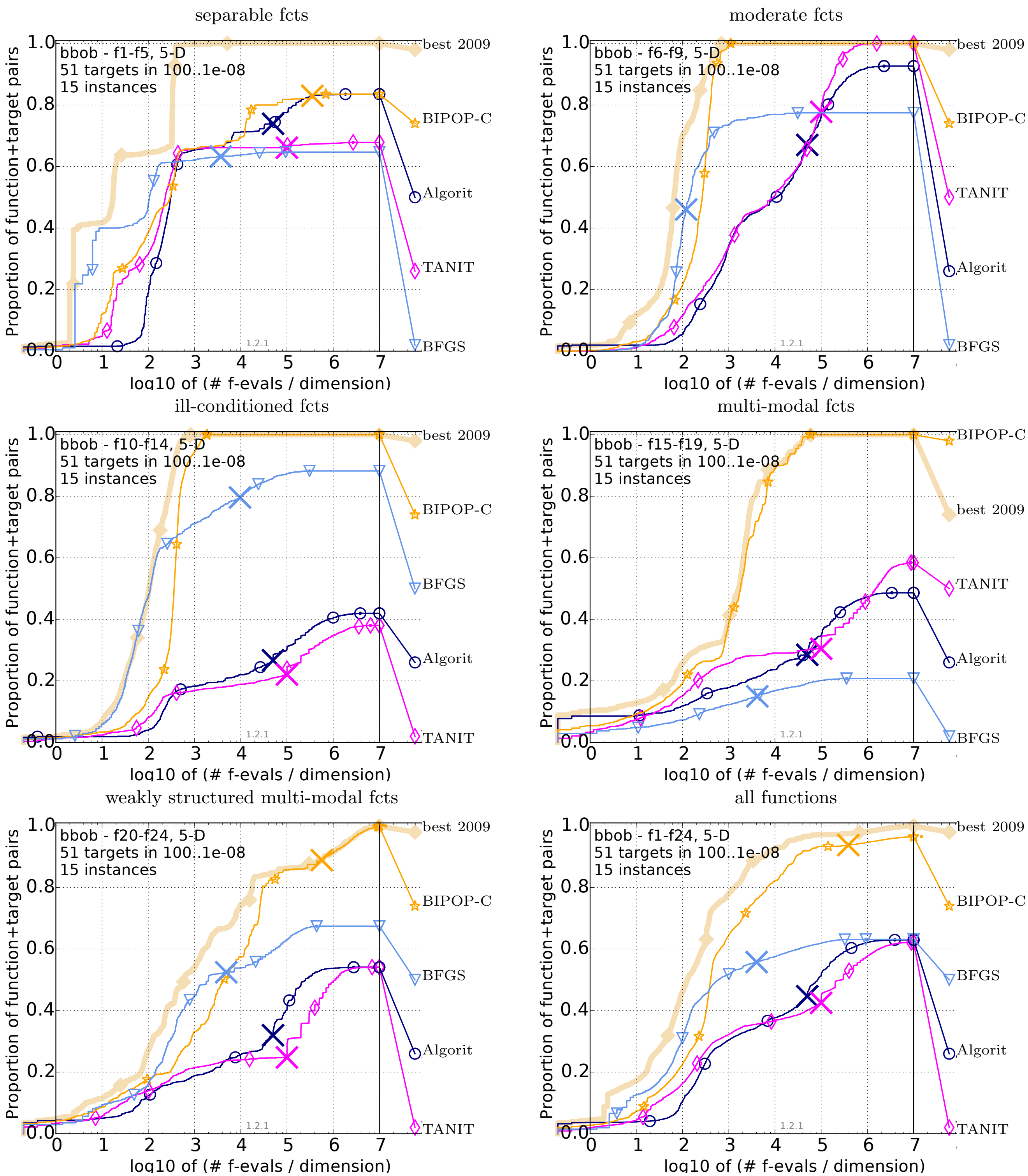

Figure 2: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D. The "best 2009" line corresponds to the best aRT observed during BBOB 2009 for each selected target.

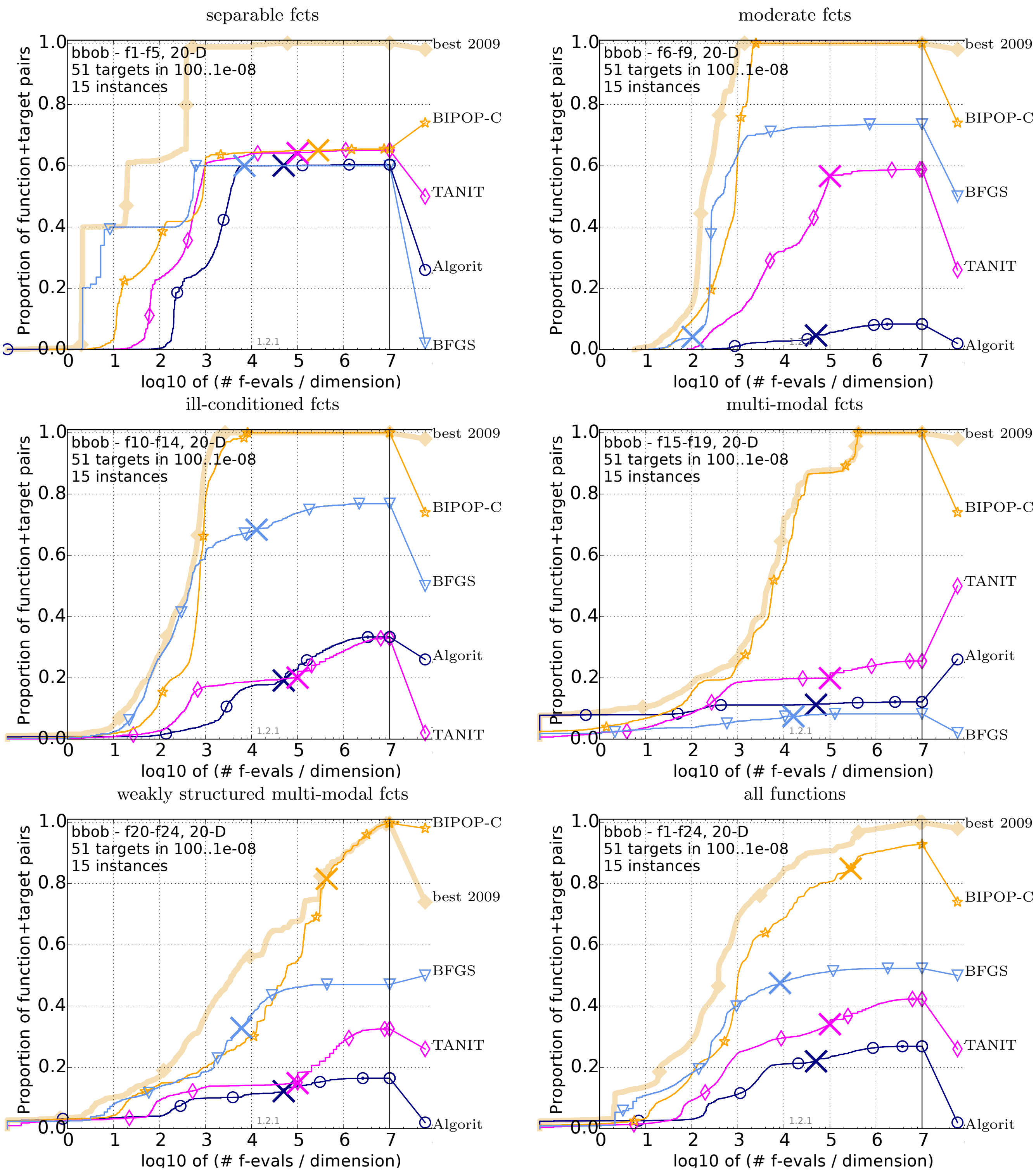

Figure 3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 20-D. The "best 2009" line corresponds to the best aRT observed during BBOB 2009 for each selected target.

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1	11	12	12	12	12	12	12	15/15
Algorit	27(17)	44(15)	55(16)	67(13)	81(10)	109(10)	134(9)	15/15
TANIT	6.2(5)	16(9)	28(6)	39(5)	47(8)	71(13)	90(8)	15/15
BIPOP-C	3.2(2)	9.0(4)	15(4)	21(4)	27(5)	40(4)	53(5)	15/15
BFGS	**1.2**(0)	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	**1.1**(0)$^{\star 4}$	15/15
f2	83	87	88	89	90	92	94	15/15
Algorit	11(2)	12(2)	14(1)	16(1)	18(0.7)	21(3)	24(3)	15/15
TANIT	10(5)	11(4)	12(5)	14(2)	15(3)	18(2)	21(3)	15/15
BIPOP-C	13(3)	16(4)	18(2)	19(1)	20(2)	21(2)	22(1)	15/15
BFGS	**3.8**(3)$^{\star 3}$	**5.6**(2)$^{\star 3}$	**6.2**(2)$^{\star 4}$	**6.5**(1)$^{\star 4}$	**6.6**(1)$^{\star 4}$	**6.9**(1)$^{\star 4}$	**7.1**(2)$^{\star 4}$	15/15
f3	716	1622	1637	1642	1646	1650	1654	15/15
Algorit	3.3(5)	47(40)	218(191)	270(292)	269(322)	269(321)	269(167)	6/15
TANIT	2.1(0.7)	852(693)	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	**1.4**(1)	**16**(16)	**139**(263)	**139**(9)	**139**(100)	**139**(46)	**140**(17)	14/15
BFGS	107(106)	∞	∞	∞	∞	∞	∞ *2e4*	0/15
f4	809	1633	1688	1758	1817	1886	1903	15/15
Algorit	3.9(3)	∞	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	97(155)	∞	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	**2.7**(2)	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BFGS	169(129)	∞	∞	∞	∞	∞	∞ *2e4*	0/15
f5	10	10	10	10	10	10	10	15/15
Algorit	36(10)	49(6)	50(18)	50(10)	50(10)	50(17)	50(8)	15/15
TANIT	6.0(2)	8.6(1)	9.4(4)	9.4(3)	9.4(5)	9.4(5)	9.4(7)	15/15
BIPOP-C	4.5(2)	6.5(1)	6.6(2)	6.6(2)	6.6(2)	6.6(2)	6.6(2)	15/15
BFGS	**1.9**(0.5)$^{\star 3}$	**3.0**(1)$^{\star 3}$	**3.1**(1)$^{\star 3}$	**3.1**(0.9)$^{\star 3}$	**3.1**(0.9)$^{\star 3}$	**3.1**(0.5)$^{\star 3}$	**3.1**(1)$^{\star 3}$	15/15
f6	114	214	281	404	580	1038	1332	15/15
Algorit	9.2(4)	7.5(3)	7.9(4)	7.2(4)	6.1(3)	4.7(1)	4.8(1)	15/15
TANIT	6.1(2)	6.0(2)	6.8(2)	6.6(2)	6.1(2)	5.1(2)	5.1(2)	15/15
BIPOP-C	**2.3**(1)	**2.1**(0.9)*	**2.2**(0.6)*	**1.9**(0.3)*	**1.7**(0.2)*	**1.3**(0.3)*	**1.3**(0.2)$^{\star 4}$	15/15
BFGS	3.0(2)	3.3(1)	3.4(1)	3.0(0.9)	2.5(1)	2.0(0.8)	7.8(12)	15/15
f7	24	324	1171	1451	1572	1572	1597	15/15
Algorit	29(24)	52(63)	115(95)	213(324)	253(202)	253(291)	675(1284)	2/15
TANIT	9.3(5)	449(773)	564(640)	592(353)	548(785)	548(162)	540(475)	6/15
BIPOP-C	**5.0**(5)	**1.5**(2)*	**1**(0.6)	**1**(0.9)	**1**(0.6)	**1**(0.5)	**1**(0.7)	15/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *600*	0/15
f8	73	273	336	372	391	410	422	15/15
Algorit	98(321)	287(438)	364(627)	494(340)	610(648)	1522(1068)	∞ *3e5*	0/15
TANIT	22(49)	701(12)	654(1128)	713(345)	817(973)	1056(1195)	1297(606)	11/15
BIPOP-C	3.2(1)	3.7(3)	4.5(3)	4.7(0.6)	4.8(1)	5.1(2)	5.4(4)	15/15
BFGS	**2.1**(1)	**1.8**(0.5)$^{\star 2}$	**1.6**(0.7)$^{\star 3}$	**1.5**(1)$^{\star 3}$	**1.5**(1)$^{\star 3}$	**1.5**(1)$^{\star 3}$	**1.5**(0.6)$^{\star 3}$	15/15
f9	35	127	214	263	300	335	369	15/15
Algorit	33(30)	355(23)	317(61)	494(741)	814(558)	3459(4383)	4994(5843)	1/15
TANIT	13(2)	375(199)	357(787)	545(403)	753(686)	1319(3185)	2523(3388)	5/15
BIPOP-C	5.8(2)	8.7(3)	7.2(5)	6.7(2)	6.4(2)	6.3(4)	6.2(2)	15/15
BFGS	**3.6**(2)*	**3.0**(2)$^{\star 2}$	**2.0**(1)$^{\star 3}$	**1.8**(0.7)$^{\star 3}$	**1.6**(0.9)$^{\star 3}$	**1.5**(0.5)$^{\star 4}$	**1.4**(0.7)$^{\star 4}$	15/15
f10	349	500	574	607	626	829	880	15/15
Algorit	1866(1543)	∞	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	2662(4557)	4643(5087)	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	3.5(0.8)	2.9(0.5)	2.7(0.4)	2.7(0.3)	2.8(0.3)	2.3(0.2)	**2.4**(0.2)$^{\star 2}$	15/15
BFGS	**1**(0.5)$^{\star 4}$	**1**(0.3)$^{\star 4}$	**1**(0.2)$^{\star 4}$	**1**(0.1)$^{\star 4}$	**1**(0.3)$^{\star 4}$	**1.1**(0.9)$^{\star 2}$	23(38)	5/15
f11	143	202	763	977	1177	1467	1673	15/15
Algorit	1.1e4(2e4)	∞	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	5.1e4(8e4)	∞	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	8.4(2)	7.2(2)	2.2(0.4)	**1.8**(0.2)	**1.6**(0.3)*	**1.4**(0.2)$^{\star 4}$	**1.3**(0.1)$^{\star 4}$	15/15
BFGS	**1**(0.1)$^{\star 4}$	**1**(0.3)$^{\star 4}$	**1.1**(1)*	1.9(3)	8.2(12)	199(283)	∞ *4e4*	0/15
f12	108	268	371	413	461	1303	1494	15/15
Algorit	1552(2888)	1072(1167)	4389(3542)	∞	∞	∞	∞ *3e5*	0/15
TANIT	5290(8087)	2897(6535)	5400(8432)	1.7e4(2e4)	∞	∞	∞ *5e5*	0/15
BIPOP-C	11(9)	7.4(0.8)	7.4(6)	7.5(4)	7.7(3)	3.3(2)	**3.3**(2)	15/15
BFGS	**1.1**(0.8)$^{\star 4}$	**1**(0.6)$^{\star 4}$	**1**(0.5)$^{\star 4}$	**1**(0.8)$^{\star 4}$	**1**(0.8)$^{\star 4}$	**2.0**(3)	49(34)	5/15
f13	132	195	250	319	1310	1752	2255	15/15
Algorit	129(121)	831(1091)	1338(2289)	5246(5751)	2675(3102)	∞	∞ *3e5*	0/15
TANIT	4328(9456)	1.7e4(2e4)	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	3.9(2)	5.4(3)	5.9(2)	5.4(1)	**1.6**(0.2)	**1.5**(0.2)$^{\star 4}$	**1.7**(0.6)$^{\star 4}$	15/15
BFGS	**1**(0.2)$^{\star 4}$	**1**(0.1)$^{\star 4}$	**1**(0.1)$^{\star 4}$	**1**(0.1)$^{\star 4}$	4.8(19)	136(251)	∞ *5e4*	0/15
f14	10	41	58	90	139	251	476	15/15
Algorit	18(19)	16(12)	16(5)	14(3)	34(22)	1625(694)	∞ *3e5*	0/15
TANIT	4.5(3)	6.6(3)	7.9(2)	7.6(2)	27(26)	1.4e4(1e4)	∞ *5e5*	0/15
BIPOP-C	**1.1**(0.9)	2.8(0.9)	3.7(0.6)	4.0(0.6)	4.6(2)	5.4(1)	**4.5**(0.3)$^{\star 4}$	15/15
BFGS	2.2(2)	**1.7**(1)	**1.8**(1)$^{\star 3}$	**1.5**(0.7)$^{\star 4}$	**1.3**(0.6)$^{\star 4}$	**1**(0.3)$^{\star 4}$	350(398)	0/15
f15	511	9310	19369	19743	20073	20769	21359	14/15
Algorit	13(4)	79(111)	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	4.2(5)	109(81)	362(607)	355(266)	349(436)	337(457)	328(533)	1/15
BIPOP-C	**1.6**(0.8)*	**1.5**(0.7)	**1.2**(0.7)	**1.2**(0.6)	**1.2**(0.7)	**1.2**(0.7)	**1.2**(0.7)	15/15
BFGS	87(103)	∞	∞	∞	∞	∞	∞ *2e4*	0/15
f16	120	612	2662	10163	10449	11644	12095	15/15
Algorit	380(600)	689(719)	419(255)	351(455)	∞	∞	∞ *3e5*	0/15
TANIT	6.8(6)	478(469)	772(450)	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	**3.0**(3)	**3.6**(3)$^{\star 3}$	**2.6**(1.0)$^{\star 3}$	**1.1**(0.8)$^{\star 4}$	**1.3**(2)$^{\star 4}$	**1.4**(1)$^{\star 4}$	**1.4**(1)$^{\star 4}$	15/15
BFGS	153(113)	960(1434)	∞	∞	∞	∞	∞ *4e4*	0/15
f17	5.2	215	899	2861	3669	6351	7934	15/15
Algorit	1.6e4(3e4)	918(1267)	291(166)	119(82)	152(241)	119(97)	96(80)	1/15
TANIT	14(8)	1.9(1)	1.1(0.3)	117(88)	375(477)	1103(1437)	∞ *5e5*	0/15
BIPOP-C	**3.4**(4)	**1**(0.5)$^{\star 2}$	**1**(2)*	**1**(0.8)	**1**(0.9)*	**1**(0.6)$^{\star 2}$	**1.2**(0.7)$^{\star 4}$	15/15
BFGS	120(59)	645(863)	∞	∞	∞	∞	∞ *2e4*	0/15
f18	103	378	3968	8451	9280	10905	12469	15/15
Algorit	397(1235)	152(250)	86(105)	423(325)	∞	∞	∞ *3e5*	0/15
TANIT	2.0(1)	97(332)	189(284)	829(1435)	∞	∞	∞ *5e5*	0/15
BIPOP-C	**1**(0.6)	**3.4**(7)*	**1**(1)	**1**(0.5)$^{\star 2}$	**1**(0.3)$^{\star 4}$	**1.2**(0.5)$^{\star 4}$	**1.3**(0.4)$^{\star 4}$	15/15
BFGS	57(109)	∞	∞	∞	∞	∞	∞ *2e4*	0/15
f19	1	1	242	1.0e5	1.2e5	1.2e5	1.2e5	15/15
Algorit	**1**(0)$^{\star 3}$	**1**(0)$^{\star 4}$	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	59(34)	2956(3937)	1.4e4(1e4)	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	20(13)	2801(1151)	**161**(168)	**1**(0.8)	**1**(0.7)	**1**(0.7)	**1**(0.9)	15/15
BFGS	1655(2126)	2.2e4(2e4)	1780(3004)	∞	∞	∞	∞ *3e4*	0/15
f20	16	851	38111	51362	54470	54861	55313	14/15
Algorit	26(17)	134(162)	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	7.3(4)	1182(3968)	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	3.3(2)	8.2(10)	**2.8**(0.8)	**2.2**(2)	**2.1**(2)	**2.2**(1)	**2.2**(0.7)	15/15
BFGS	**1.8**(1)	**2.5**(2)	10(9)	7.6(7)	7.2(13)	7.1(5)	7.1(6)	1/15
f21	41	1157	1674	1692	1705	1729	1757	14/15
Algorit	76(101)	199(325)	218(215)	216(278)	214(303)	212(130)	209(225)	7/15
TANIT	5.1(3)	865(865)	822(1195)	813(1625)	807(807)	797(1013)	784(854)	4/15
BIPOP-C	**2.3**(1)	14(28)	24(73)	25(120)	25(63)	25(4)	25(35)	15/15
BFGS	3.8(5)	**1.4**(2)	**1.9**(3)	**1.9**(3)	**1.9**(2)	**1.9**(1)	**2.0**(2)	15/15
f22	71	386	938	980	1008	1040	1068	14/15
Algorit	18(25)	400(539)	571(613)	551(819)	539(505)	531(495)	529(1117)	5/15
TANIT	1088(3)	2592(5178)	3468(5731)	3320(2168)	3230(4464)	3137(2043)	3064(5269)	2/15
BIPOP-C	6.9(15)	20(53)	45(55)	43(24)	42(87)	41(70)	40(92)	15/15
BFGS	**3.1**(3)	**2.9**(4)	**2.1**(1)	**2.1**(2)	**2.0**(1)	**2.0**(1)	**2.6**(0.8)	14/15
f23	3.0	518	14249	27890	31654	33030	34256	15/15
Algorit	7.4(14)	92(133)	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	4.2(1)	100(101)	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	**1.7**(2)	**13**(23)	**3.7**(4)	**2.1**(0.9)	**1.8**(2)	**1.8**(1)	**1.8**(0.9)	15/15
BFGS	11(14)	31(94)	∞	∞	∞	∞	∞ *2e4*	0/15
f24	1622	2.2e5	6.4e6	9.6e6	9.6e6	1.3e7	1.3e7	3/15
Algorit	37(35)	∞	∞	∞	∞	∞	∞ *3e5*	0/15
TANIT	10(12)	∞	∞	∞	∞	∞	∞ *5e5*	0/15
BIPOP-C	**2.1**(2)	**1.6**(3)	**1**(2)	**1**(1)	**1**(0.9)	**1**(1)	**1**(1.0)	3/15
BFGS	69(69)	∞	∞	∞	∞	∞	∞ *2e4*	0/15

Table 1: Average running time (aRT in number of function evaluations) divided by the respective best aRT measured during BBOB-2009 in dimension 5. The aRT and in braces, as dispersion measure, the half difference between 10 and 90%-tile of bootstrapped run lengths appear for each algorithm and target, the corresponding best aRT in the first row. The different target Δf-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\mathrm{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p = 0.05$ or $p = 10^{-k}$ when the number k following the star is larger than 1, with Bonferroni correction of 110. A ↓ indicates the same tested against the best algorithm of BBOB-2009. Best results are printed in bold.

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1	43	43	43	43	43	43	43	15/15
Algorit	120(68)	241(60)	397(82)	573(102)	745(174)	1078(161)	1436(156)	15/15
TANIT	32(3)	58(6)	84(7)	111(8)	138(7)	195(10)	249(18)	15/15
BIPOP-C	7.9(2)	14(3)	20(3)	26(3)	33(2)	45(2)	57(2)	15/15
BFGS	**1**(0)$^{\star 4}$	**1**(0)$^{\star 4}$	**1**(0)$^{\star 4}$	**1**(0)$^{\star 4}$	**1**(0)$^{\star 4}$	**1**(0)$^{\star 4}$	**1**(0)$^{\star 4}$	15/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f2	385	386	387	388	390	391	393	15/15
Algorit	82(9)	99(17)	116(29)	135(32)	156(21)	197(25)	238(26)	15/15
TANIT	23(5)	26(4)	29(3)	32(5)	35(5)	41(3)	47(4)	15/15
BIPOP-C	35(7)	40(5)	44(3)	45(2)	47(3)	48(2)	50(2)	15/15
BFGS	**20**(3)	**24**(1.0)	**26**(4)	**27**(5)$^{\star 2}$	**27**(3)$^{\star 3}$	**28**(4)$^{\star 4}$	**28**(4)$^{\star 4}$	15/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f3	5066	7626	7635	7637	7643	7646	7651	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	57(12)	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	**12**(8)$^{\star 2}$	∞	∞	∞	∞	∞	∞ *6e6*	0/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *1e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	∞	∞	∞	∞	∞	∞	∞ *6e6*	0/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *2e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f5	41	41	41	41	41	41	41	15/15
Algorit	97(16)	111(27)	112(13)	112(6)	112(37)	112(30)	112(38)	15/15
TANIT	23(3)	29(2)	30(5)	31(5)	31(4)	31(6)	31(6)	15/15
BIPOP-C	5.1(0.4)	6.2(1)	6.3(1)	6.3(1)	6.3(0.9)	6.3(1)	6.3(0.9)	15/15
BFGS	**2.4**(0.5)$^{\star 4}$	**2.7**(0.3)$^{\star 4}$	**2.8**(0.4)$^{\star 4}$	**2.8**(0.4)$^{\star 4}$	**2.8**(0.5)$^{\star 4}$	**2.8**(0.3)$^{\star 4}$	**2.8**(0.5)$^{\star 4}$	15/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f6	1296	2343	3413	4255	5220	6728	8409	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	7.8(2)	8.0(2)	8.3(4)	9.4(2)	10(5)	11(4)	12(6)	15/15
BIPOP-C	**1.5**(0.3)$^{\star 3}$	**1.3**(0.3)$^{\star 4}$	**1.2**(0.2)$^{\star 4}$	**1.1**(0.2)$^{\star 4}$	**1.1**(0.2)$^{\star 4}$	**1.2**(0.1)$^{\star 4}$	**1.2**(0.1)$^{\star 4}$	15/15
BFGS	3.6(0.8)	3.5(2)	3.4(0.7)	3.5(0.8)	3.5(1.0)	3.6(0.6)	45(34)	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f7	1351	4274	9503	16523	16524	16524	16969	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	1119(1751)	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	**1**(0.2)$^{\star 4}$	**4.9**(3)	**3.5**(0.6)	**2.2**(0.3)	**2.2**(0.2)	**2.2**(0.3)	**2.1**(0.2)	15/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *2100*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f8	2039	3871	4040	4148	4219	4371	4484	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	102(6)	126(6)	160(4)	213(3)	273(3)	393(2)	∞ *2e6*	0/15
BIPOP-C	4.0(1)	4.0(0.8)	4.3(0.6)	4.5(0.6)	4.5(1)	4.6(0.9)	4.6(0.7)	15/15
BFGS	**1.8**(0.2)$^{\star 4}$	**1.2**(0.1)$^{\star 4}$	**1.2**(0.2)$^{\star 4}$	**1.2**(0.2)$^{\star 4}$	**1.2**(0.2)$^{\star 4}$	**1.2**(0.1)$^{\star 4}$	**1.2**(0.1)$^{\star 4}$	15/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f9	1716	3102	3277	3379	3455	3594	3727	15/15
Algorit	315(99)	765(900)	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	166(12)	331(327)	482(315)	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	4.7(2)	5.7(5)	6.0(0.7)	6.1(4)	6.1(3)	6.1(4)	6.1(0.8)	15/15
BFGS	**2.2**(0.4)$^{\star 4}$	**2.2**(1.0)$^{\star 4}$	**2.1**(1)$^{\star 4}$	**2.1**(0.9)$^{\star 4}$	**2.0**(0.8)$^{\star 4}$	**2.0**(1)$^{\star 4}$	**1.9**(0.8)$^{\star 4}$	15/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f10	7413	8661	10735	13641	14920	17073	17476	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	1.9(0.2)	1.8(0.1)	1.6(0.1)	1.3(0.1)	1.2(0.0)	**1.1**(0.0)	**1.1**(0.0)$^{\star 4}$	15/15
BFGS	**1.0**(0.1)$^{\star 4}$	**1**(0.1)$^{\star 4}$	**1**(0.5)*	**1.1**(0.5)	**1.1**(0.4)	3.1(7)	∞ *1e6*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f11	1002	2228	6278	8586	9762	12285	14831	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	10(0.7)	5.1(0.3)	1.9(0.1)	**1.5**(0.0)	**1.4**(0.0)$^{\star 4}$	**1.2**(0.0)$^{\star 4}$	**1.0**(0.0)$^{\star 4}$	15/15
BFGS	**1**(0.5)$^{\star 4}$	**1**(0.8)$^{\star 4}$	**1.3**(2)$^{\star 2}$	2.6(3)	147(87)	∞	∞ *2e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f12	1042	1938	2740	3156	4140	12407	13827	15/15
Algorit	51(8)	112(6)	1031(1096)	1296(1111)	3404(2476)	1138(1793)	∞ *1e6*	0/15
TANIT	489(960)	1554(2064)	2924(4562)	8877(5229)	6766(4106)	∞	∞ *2e6*	0/15
BIPOP-C	3.0(1.0)	4.0(4)	4.5(4)	4.9(3)	4.5(3)	1.9(0.6)	**2.0**(0.6)$^{\star 3}$	15/15
BFGS	**1.6**(0.3)$^{\star 2}$	**1.6**(0.9)	**1.6**(0.5)	**1.7**(1)$^{\star 2}$	**1.6**(0.7)$^{\star 2}$	**1.8**(2)	45(40)	1/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f13	652	2021	2751	3507	18749	24455	30201	15/15
Algorit	102(37)	131(148)	737(717)	1253(642)	377(196)	∞	∞ *1e6*	0/15
TANIT	781(1537)	1986(2969)	4733(2909)	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	4.3(4)	2.7(0.1)	5.1(6)	6.2(5)	**1.5**(0.8)	**2.3**(2)$^{\star 4}$	**3.0**(2)$^{\star 4}$	15/15
BFGS	**1.7**(0.3)$^{\star 3}$	**1**(0.0)$^{\star 2}$	**1**(0.0)$^{\star 2}$	**1**(0.0)	23(43)	∞	∞ *5e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f14	75	239	304	451	932	1648	15661	15/15
Algorit	1003(3339)	2850(4179)	2279(1668)	∞	∞	∞	∞ *1e6*	0/15
TANIT	18(4)	13(1)	15(2)	17(3)	49(12)	∞	∞ *2e6*	0/15
BIPOP-C	3.9(1)	2.9(0.5)	3.7(0.3)	4.3(0.3)	4.1(0.3)	6.2(0.6)	**1.2**(0.1)$^{\star 4}$	15/15
BFGS	**2.7**(2)	**1.8**(0.6)$^{\star 3}$	**2.0**(0.6)$^{\star 4}$	**1.8**(0.5)$^{\star 4}$	**1.2**(0.3)$^{\star 4}$	**1.1**(0.2)$^{\star 4}$	∞ *2e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f15	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	**1**(0.4)$^{\star 4}$	**2.0**(1)	**1.4**(0.5)	**1.4**(0.4)	**1.4**(0.5)	**1**(0.3)	**1**(0.3)	15/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *1e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f16	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
Algorit	1.0e4(9395)	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	**1.7**(0.5)$^{\star 4}$	**1.0**(0.5)$^{\star 4}$	**1.2**(0.7)$^{\star 4}$	**1**(0.8)$^{\star 4}$	**1**(0.2)$^{\star 4}$	**1**(0.6)$^{\star 4}$	**1**(0.7)$^{\star 4}$	15/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *3e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f17	63	1030	4005	12242	30677	56288	80472	15/15
Algorit	105(349)	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	15(5)	4.3(1)	38(0.9)	110(245)	261(147)	∞	∞ *2e6*	0/15
BIPOP-C	**2.2**(1)$^{\star 4}$	**1**(0.3)$^{\star 4}$	**1**(1)$^{\star 2}$	**1**(0.8)	**1.2**(1)	**1.3**(0.5)$^{\star 4}$	**1.4**(1)$^{\star 4}$	15/15
BFGS	359(489)	∞	∞	∞	∞	∞	∞ *4e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f18	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	4.1(0.8)	**2.0**(0.7)	205(281)	981(1103)	∞	∞	∞ *2e6*	0/15
BIPOP-C	**1.0**(0.3)$^{\star 4}$	2.4(7)	**1.2**(1.0)	**1.6**(1)$^{\star 3}$	**1.1**(0.8)$^{\star 4}$	**1.7**(0.8)$^{\star 4}$	**1.6**(0.2)$^{\star 4}$	15/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *4e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f19	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
Algorit	**1**(0)$^{\star 4}$	**1**(0)$^{\star 4}$	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	750(150)	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	169(48)	2.4e4(1e4)	**1.2**(0.6)	**1**(0.3)	**1**(0.3)	**1**(0.3)	**1**(0.3)	15/15
BFGS	1.2e6(2e6)	∞	∞	∞	∞	∞	∞ *2e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f20	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
Algorit	92(86)	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	17(3)	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	4.3(1)	9.2(4)	**1**(0.6)	**1**(0.7)	**1**(0.8)	**1**(0.3)	**1**(1)	14/15
BFGS	**2.1**(0.4)$^{\star 4}$	**5.8**(4)	∞	∞	∞	∞	∞ *4e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f21	561	6541	14103	14318	14643	15567	17589	15/15
Algorit	944(1402)	679(263)	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	1785(2673)	1988(2217)	922(709)	909(698)	889(956)	836(1221)	740(597)	2/15
BIPOP-C	3.2(3)	55(133)	48(21)	47(152)	46(52)	43(23)	39(201)	13/15
BFGS	**1.9**(1)	**5.5**(6)	**4.6**(7)	**4.6**(6)	**4.5**(5)	**4.3**(2)	**7.3**(6)	2/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f22	467	5580	23491	24163	24948	26847	1.3e5	12/15
Algorit	1335(1673)	789(978)	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	2862(7499)	5018(7258)	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	6.8(10)	13(7)	215(189)	209(160)	202(134)	188(247)	37(28)	5/15
BFGS	**2.5**(2)	**1.8**(3)	**8.1**(3)	**7.9**(7)	**7.7**(5)	**10**(14)	**14**(15)	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f23	3.2	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
Algorit	10(22)	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	20(19)	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	**4.3**(4)	**32**(16)	**1**(0.8)	**1.7**(1)	**2.0**(1)	**1.2**(1.0)	**1.2**(0.9)	15/15
BFGS	47(92)	304(269)	∞	∞	∞	∞	∞ *1e5*	0/15

Δf_opt	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f24	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
Algorit	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/15
TANIT	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/15
BIPOP-C	**1**(0.7)	**1**(0.5)	**1**(0.5)	**1**(1)	**1**(0.4)	**1**(0.8)	**1**(1)	3/15
BFGS	∞	∞	∞	∞	∞	∞	∞ *1e5*	0/15

Table 2: Average running time (aRT in number of function evaluations) divided by the respective best aRT measured during BBOB-2009 in dimension 20. The aRT and in braces, as dispersion measure, the half difference between 10 and 90%-tile of bootstrapped run lengths appear for each algorithm and target, the corresponding best aRT in the first row. The different target Δf-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_\mathrm{opt} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p = 0.05$ or $p = 10^{-k}$ when the number k following the star is larger than 1, with Bonferroni correction of 110. A ↓ indicates the same tested against the best algorithm of BBOB-2009. Best results are printed in bold.

continuous optimizers in a black-box setting. *ArXiv e-prints*,

[6] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009.

[7] K. Price. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pages 153–157, 1997.

[8] R. Ros. Benchmarking the bfgs algorithm on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2409–2414. ACM, 2009.

[9] T. Tušar, D. Brockhoff, N. Hansen, and A. Auger. COCO: The bi-objective black-box optimization benchmarking (bbob-biobj) test suite. *ArXiv e-prints*,