(1,λ)ES with AII adaptation scheme for Black-Box Optimization

Benchmarking for Noiseless Functions (bbob suite)

Eden Belouadah; Taycir Yahmed; Mohamed Ali Darghouth; Ghiles Sidi Said; Amine Biad;

ABSTRACT

An implementation of the AII algorithm [6] is proposed in this paper for continuous optimization problems. This paper is organized as follows: We first introduce the interest of this project and we present the COCO platform in which we are going to integrate and execute the AII algorithm. Furthermore, we present the algorithm and give a pseudo-code for it then we conclude by the presentation, the discussion of the results obtained and a comparison with BIPOP-CMA-ES and BFGS algorithms.

KEYWORDS

AII Algorithm, (1,λ)ES, Benchmarking, Black-box optimization, Continuous optimization, COCO platform, BBOB suite

ACM Reference format:
Eden Belouadah; Taycir Yahmed; Mohamed Ali Darghouth; Ghiles Sidi Said; Amine Biad; . 2017. (1,λ)ES with AII adaptation scheme for Black-Box Optimization . In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 9 pages.
DOI: 10.475/123_4

1 INTRODUCTION

With technology evolution, problems become more and more complex. This lead researchers to search for optimization issues to solve them. There are so many types of optimization problems. We are interested in this project in blackbox continuous optimization ones. For this purpose, we try to use our knowledge that we learned from the optimization lectures and to implement the AII algorithm proposed by Hansen et al.[6] hoping that it will give a good results on benchmark functions. The algorithm's performance will be compared with two other algorithms and it will be discussed by analyzing the ECDF plots.

1.1 COCO platform

COCO (COmparing Continuous Optimizers) is a platform which help to automatically compare continuous optimization algorithms. It presents many suites of continuous problems (for both constrained and unconstrained optimization):

- bbob suite: contains 24 noiseless functions with more than 140 datasets provided by different algorithms,
- bbob-noisy suite: contains 30 noisy functions with more than 40 datasets provided by different algorithms,
- bbob-biobj suite: contains 55 bi-objective functions with more than 16 datasets provided by different algorithms.

GECCO '17, Berlin, Germany

DOI: 10.475/123_4

1.2 bbob suite

AII algorithm works on mono-objective functions, this leads us to work on bbob suite. Tests will be done for these dimensions: 2, 3, 5, 10, 20 and 40. We have to mention that bbob suite functions are locally perturbed by non-linear transformations.

2 ALGORITHM PRESENTATION

Before we explain the AII algorithm, we should talk briefly about (1,λ)ES algorithm in which the AII algorithm will be integrated. We have to point out that AII adaptation is proposed by its author to fit all ($\mu/\mu, \lambda$)ES but we choose to use it with (1,λ)ES as the author did.

(1,λ)ES is an evolutionary algorithm where $\mu = 1$, this means that at every iteration, only one parent (the best one) from the offspring population is chosen to be the solution with which we start the next iteration.

The AII algorithm, or Derandomized Adaptation Of n individual step sizes and one direction is an adaptation proposed by Nikolaus Hansen and al.[6] for the arbitrary normal mutation distributions in Evolution strategies. It's an extension of the derandomized individual step size adaptation proposed by Ostermeier et al. [8]. In AII, the reproduction scheme is based on two important aspects: n individual step sizes and the adapted direction. The latter is done by accumulating every time the selected mutation steps in the generation sequence.

In AII, the mutation is done by adding a vector from n variable-independent Gaussian distributions (as isodensity surfaces, a.k.a. surfaces with same density) plus a vector $\mathbf{r}$ normally distributed along a specific direction. The resulting mutation distribution is not independent of the coordinating system.

Despite of the fact that AII doesn't produce arbitrary mutation distribution (not every normal distribution can be produced), it operates well on functions which need correlated mutations for reasonable progress [6].

The pseudo-code for the algorithm is given in Algorithm 1.

IMPLEMENTATION OF THE ALGORITHM

The parameters that we chose are the same used by the author of AII as following:

- n: the dimension of the problem,
- $\lambda = 10$: the size of the offspring,
- $c = \sqrt{1/n}$,
- $c_r = 3/(n+3)$,
- $cu = \sqrt{(2-c)/c}$
- $\beta = 1/n$,
- $\beta_{ind} = 1/(4n)$,
- $\beta_r = \sqrt{1/(4n)}$,
- $\widehat{\chi_n} = \sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2})$,
- $\widehat{\chi_1} = \sqrt{2/\pi}$,

Data: ubounds, lbounds, f, budget, target
Result: The best solution X_{min} with fitness f_{min}
Initialization of $n, \lambda, c, \beta, \widehat{\chi_n}, \widehat{\chi_1}, cr, \beta_{ind}, \beta_r, cu, X_{min}$;
$f_{min}, \delta, \delta_r, \mathbf{s}, sr, \mathbf{r}, \mathbf{z}, z_r$;
while *budget≥0* **do**
  X_{parent}=Copy(X_{min});
  for *k=1,...,λ* **do**
    Initialize z_r^k, z^k;
    for *i=1,...,n* **do**
      $x_i^{N_k} = x_i^{parent} + \delta_i^{parent} z_i^k + \delta_r^{parent} z_r^k r_i^{parent}$;
    end
    $f_{current}$=f(X^{N_k});
    if $f_{current} \leq f_{min}$ **then**
      $X_{min} = X^{N_k}$;
      $z_r = z_r^k$;
      $z = z^k$;
      $f_{min} = f_{current}$
    end
  end
  Updating the parameters: $\delta, \delta_r, \mathbf{s}, sr, \mathbf{r}$;
  Budget-=λ;
end

Algorithm 1: $(1, \lambda)$ES with AII adaptation scheme

- $r = \overrightarrow{0} \in R^n$,
- $\delta = 1 \in R^n$,
- $\delta_r = 1$,
- $s = \overrightarrow{0} \in R^n$,
- $s_r = 0$,

The other parameters are initialized and eventually change their values inside of the algorithm.

The stopping criterion depends on the budget (number of function evaluations x dimension) that the user defines for the algorithm. This latter tries to optimize the objective function until no budget is remaining.

Timing experiment

In order to evaluate the CPU timing of the algorithm, we have run the AII algorithm on the bbob test suite [4] with restarts for a maximum budget equal to $9999xD$ function evaluations according to [7] (We tried $1000xD$ function evaluations also but we present here the results that we got with the maximum budget only, the results for the other budget are available on GitHub).

The Python code was run on an Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz. The time per function evaluation for dimensions 2, 3, 5, 10, 20, 40 equals $1.4e-04$,1.3e-04, $1.5e-04$, $1.5e-04$, $1.5e-04$ and $1.9e-04$ seconds respectively.

3 DISCUSSION OF RESULTS

Results of AII from experiments according to [7] and [2] on the benchmark functions given in [1, 5] are presented in Figures 1, 2 and in Tables 1, 2. The experiments were performed with COCO [3], version 0.0.0, the plots were produced with the same version.

For the rest of the paper, we will call the best algorithm from BBOB 2009, BEST_BBOB.

From Figure 1, we unfortunately see, that for functions: $f1$, $f2$, $f3$, $f4$, $f5$, $f10$, $f11$, $f13$ and $f19$, AII algorithm takes more time to reach the easiest target comparing to BEST_BBOB to reach the hardest one.

For the 6th function , we see that the results of the two algorithms are so closed, BEST_BBOB algorithm remains better.

For the functions $f7$, $f8$, $f9$, $f12$, $f14$, $f15$, $f16$, $f17$, $f18$, $f21$, $f22$, $f23$ and $f24$ we see that sometimes AII algorithm is better than Best_BBOB but just in case where AII has easier targets to reach.

From the ECDFs graphs (Figure 2), we see that AII algorithm is more robust for the functions $f1$, $f2$, $f5$, $f6$, $f12$, function because for almost all dimensions, the proportion of runs where the algorithm reaches the target in less than 4($log_{10}(\#f\_eval/D)$) is high. For example, for the 1st function (Sphere), only 2.2 ($log_{10}(\#f\_eval/D)$) are needed to reach the target.

AII algorithm starts to have some difficulties with some functions like $f7$, $f10$, $f11$ and $f16$. Indeed, it needs more function evaluations to reach the optimum, but it's still efficient anyway.

For the rest of functions (like $f23$ and $f24$), only less that 40% of runs are successful with 6 ($log_{10}(\#f\_eval/D)$). These functions are so difficult to optimize.

Of course, the AII algorithm (and any other optimization algorithm) is efficient for little dimensions and starts to have difficulties to reach the optimum with bigger dimensions. For example, for the 11th three dimensional function, 80% of runs are successful with 6($log_{10}(\#f\_eval/D)$). Whatever, for the 10D same function, only less than 40% of runs are successful with the same number of $log_{10}(\#f\_eval/D)$!.

If we take the 24th function, we find that it's one of the most difficult functions for our algorithm to optimize. even for the 2D version of the function and the highest value of $log_{10}(\#f\_eval/D)$, only 30% of runs are successful.

The Table 1 represents the average running time (in number of function evaluations) divided by the average run time of the best algorithm from BBOB 2009 in dimension 5.The Table 2 represents the same thing for the dimension 20.

The general (and obvious) thing that we can extract from these two tables is that the more we decrease the tolerated difference between the best solution found by the algorithm and the real solution of the function, the longer the algorithm takes. For example, for the 7th function, when we fix Δf to 10, the task is so easy and the algorithm converges to the optimum in only 24 #f_eval/best_#f_eval, but when we fix it to 1e-7, the task is more difficult here and the algorithm needs more time to reach the optimum, so it does in 1597 #f_eval/best_#f_eval.

When the problem is simple (in case of separable, uni-modal functions), the algorithm is always efficient whatever the stopping criteria is. This is the case of the 1st function and the 5th one for example.

We can also see that in most cases, the proportion of successful runs is 15/15=100% except for some difficult objective functions like the 24th one (here we have 3/15 successful runs only).

The last thing that we can point it out (already known thing) is that the algorithm takes more time to optimize functions with

20D comparing them to 5D ones. For example, the optimum for the 3rd function with $\Delta f = 1e-7$ in the Table 1 is reached after 1654 #f_eval and it's reached in the Table 2 with the same Δf in 7651 #f_eval/best_#f_eval.

4 AII VS. BIPOP-CMA-ES AND BFGS

The comparison tables are shown in Figures 3, 4 and 5 which represent respectively the ECDF graphs for 10D, 20D and 40D functions for the three algorithms AII, BIPOP-CMA-ES and BFGS.

For the three dimensions, we see that AII is always the worst or the second worst algorithm.

- 10D functions (Figure 3): For the functions $f1$, $f2$ and $f5$, we see that the three algorithms are close in terms of efficiency, but BFGS remains better.

  For the functions $f3$ and $f4$, the three algorithms are close to each others. BIPOP-CMA-ES is the best but it's far enough from BEST_BBOB.

  The behaviour of AII algorithm is in most of times similar to those of BFGS and BIPOP-CMA-ES
- 20D functions (Figure 4): We see the same thing as for 10D functions, AII performance is very good comparing to the two algorithms even if it's always the worst or the second worst one.

  Unfortunately, for the 22th function, AII performs very bad, only 20% of runs are successful with a budget of $6(log_{10}(\#f\_eval/D))$ while the three other algorithms reach 100% of successful runs with the same budget.
- 40D functions (Figure 5): Even with high-dimensional problems, AII is still comparable to BFGS and BIPOP-CMA-ES for separable and uni-modal functions. For the multi-modal and ill-conditioned problems AII starts to have difficulties to find the optimum though it's sometimes the second worst algorithm.

5 CONCLUSION

In this paper, we show a simple implementation of the AII algorithm. Possible improvements would be to run the experiments with a higher budget to give the algorithm the chance to optimize multi-modal, high dimensional and non-separable functions. Another potential improvement would be optimizing the algorithm more (We can think of stagnation as a stopping criterion) and why not try to tune the parameters by ourselves instead of using those used by the original author.

REFERENCES

[1] 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.

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

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

[4] 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/

[5] 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://coco.gforge.inria.fr/bbob2012-downloads Updated February 2010.

[6] Nikolaus Hansen, Andreas Ostermeier, and Andreas Gawelczyk. 1995. On the Adaptation of Arbitrary Normal Mutation Distributions in Evolution Strategies: The Generating Set Adaptation.. In *ICGA*. 57–64.

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

[8] Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. 1994. Step-size adaptation based on non-local use of selection information. *Parallel Problem Solving from NaturefiPPSN III* (1994), 189–198.

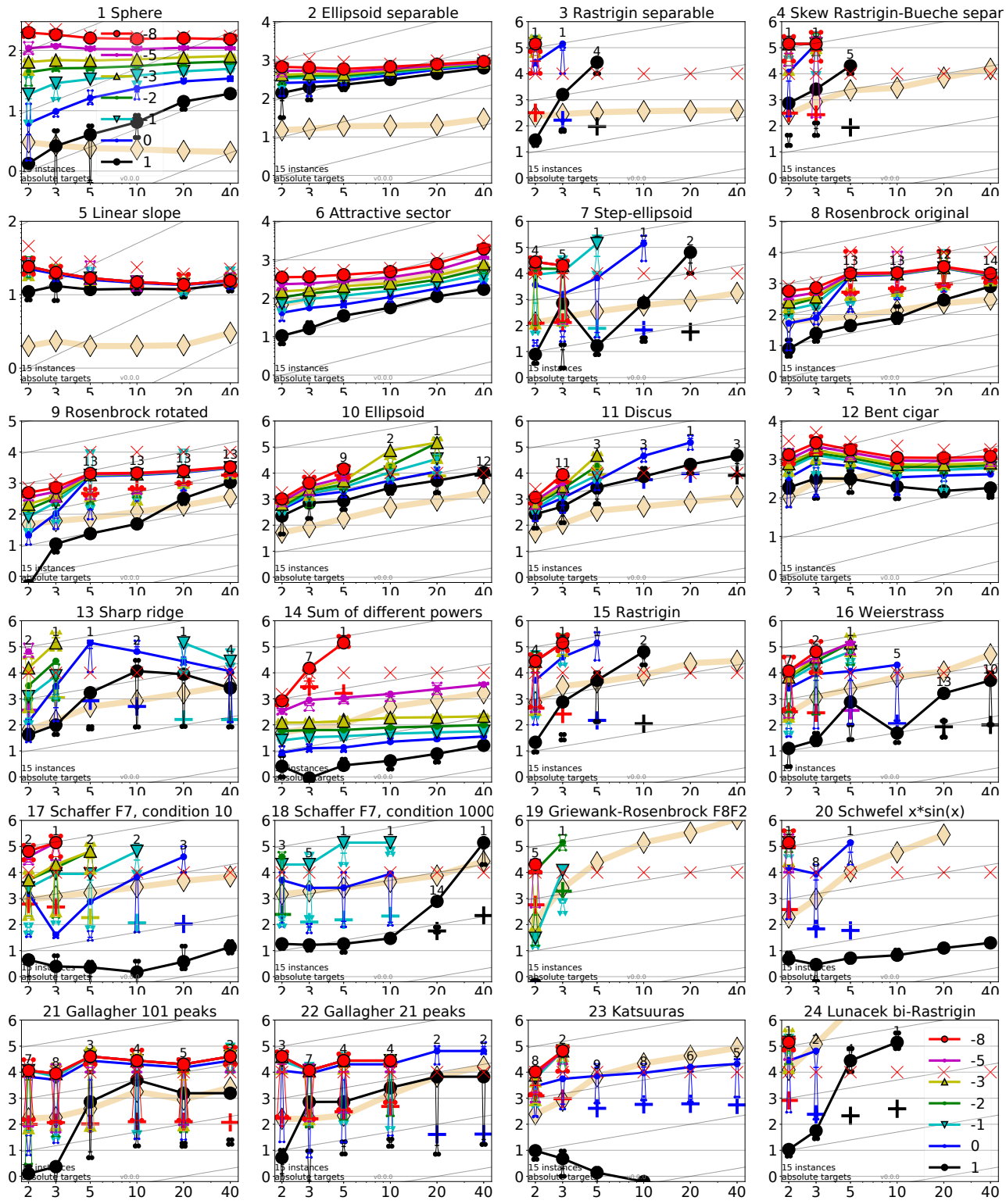

Figure 1: Scaling of runtime with dimension to reach certain target values Δf. Lines: average runtime (aRT); Cross (+): median runtime of successful runs to reach the most difficult target that was reached at least once (but not always); Cross (×): maximum number of f-evaluations in any trial. Notched 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.

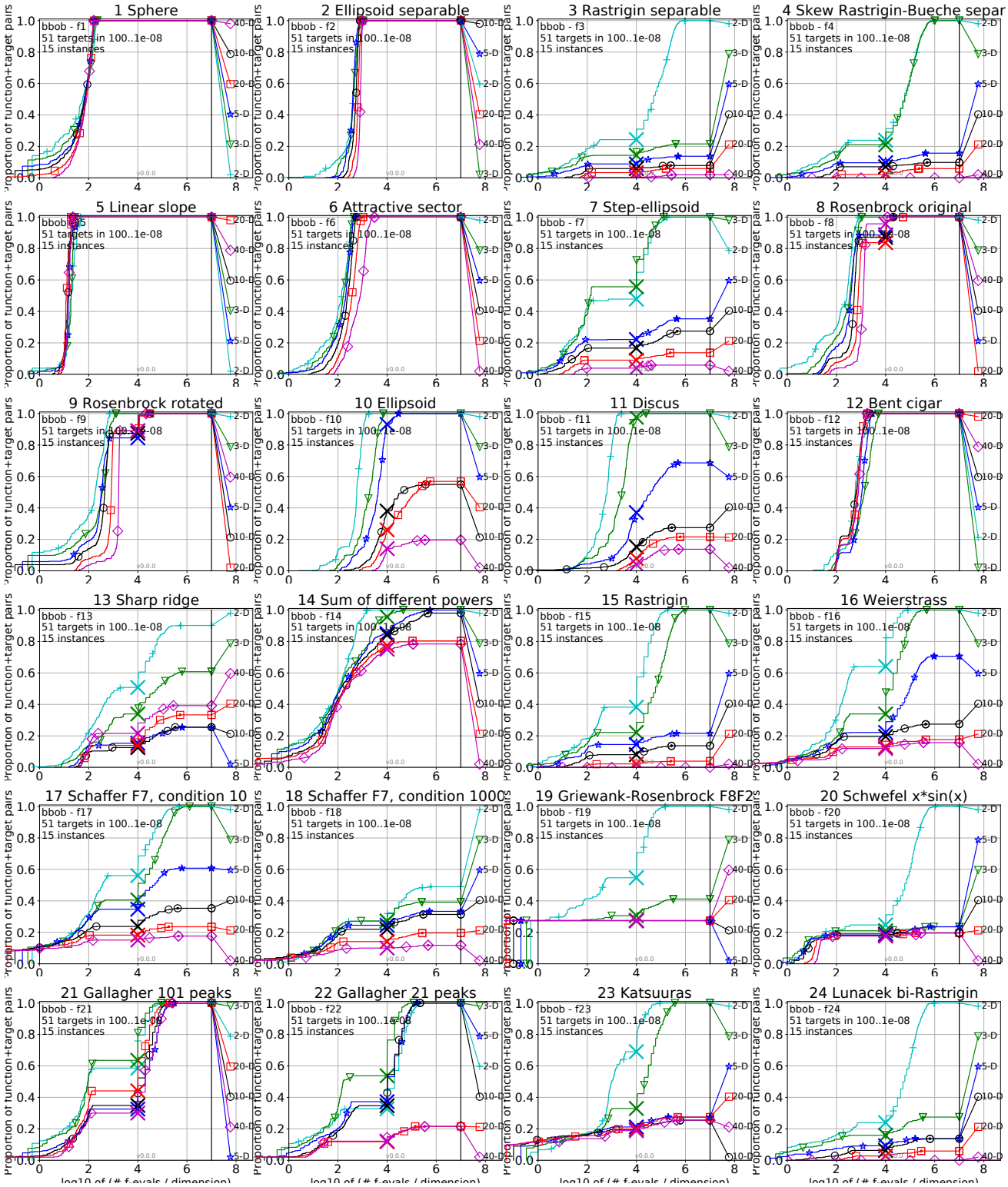

Figure 2: Empirical cumulative distribution functions (ECDF), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Lft subplots: ECDF of the number of function evaluations (FEvals) divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. The thick red line represents the most difficult target value $f_{\mathrm{opt}} + 10^{-8}$. 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
	1.8(2)	6.8(3)	14(5)	21(4)	28(6)	44(7)	60(2)	15/15
f_2	83	87	88	89	90	92	94	15/15
	14(6)	17(6)	20(3)	22(5)	24(7)	27(8)	29(7)	15/15
f_3	716	1622	1637	1642	1646	1650	1654	15/15
	193(279)	∞	∞	∞	∞	∞	∞*5.0e4*	0/15
f_4	809	1633	1688	1758	1817	1886	1903	15/15
	124(139)	∞	∞	∞	∞	∞	∞*5.0e4*	0/15
f_5	10	10	10	10	10	10	10	15/15
	5.9(0.6)	8.0(4)	8.4(0.7)	8.4(2)	8.4(2)	8.4(2)	8.4(2)	15/15
f_6	114	214	281	404	580	1038	1332	15/15
	1.5(0.6)	1.6(0.5)	2.1(0.4)	2.0(0.4)	1.8(0.3)	1.3(0.2)	1.4(0.2)	15/15
f_7	24	324	1171	1451	1572	1572	1597	15/15
	3.4(3)	104(155)	598(608)	∞	∞	∞	∞*5.0e4*	0/15
f_8	73	273	336	372	391	410	422	15/15
	3.0(2)	31(91)	27(38)	25(68)	25(161)	24(32)	25(32)	13/15
f_9	35	127	214	263	300	335	369	15/15
	3.4(0.9)	66(4)	41(2)	35(144)	31(125)	29(40)	27(36)	13/15
f_{10}	349	500	574	607	626	829	880	15/15
	12(14)	20(9)	25(14)	30(26)	40(30)	40(10)	54(23)	9/15
f_{11}	143	202	763	977	1177	1467	1673	15/15
	98(59)	129(60)	54(25)	96(182)	204(159)	∞	∞*5.0e4*	0/15
f_{12}	108	268	371	413	461	1303	1494	15/15
	15(13)	12(9)	12(5)	13(8)	14(8)	5.8(3)	5.7(2)	15/15
f_{13}	132	195	250	319	1310	1752	2255	15/15
	65(198)	3611(6859)	∞	∞	∞	∞	∞*5.0e4*	0/15
f_{14}	10	41	58	90	139	251	476	15/15
	1.4(1)	1.7(0.8)	3.1(0.8)	3.5(0.5)	5.0(2)	22(9)	254(400)	1/15
f_{15}	511	9310	19369	19743	20073	20769	21359	14/15
	50(98)	75(184)	∞	∞	∞	∞	∞*5.0e4*	0/15
f_{16}	120	612	2662	10163	10449	11644	12095	15/15
	31(2)	94(245)	122(117)	69(80)	67(89)	60(77)	∞*5.0e4*	0/15
f_{17}	5.0	215	899	2861	3669	6351	7934	15/15
	2.3(3)	18(58)	49(56)	114(118)	89(177)	∞	∞*5.0e4*	0/15
f_{18}	103	378	3968	8451	9280	10905	12469	15/15
	0.88(0.6)	34(66)	177(117)	∞	∞	∞	∞*5.0e4*	0/15
f_{19}	1	1	242	1.0e5	1.2e5	1.2e5	1.2e5	15/15
	1	1	∞	∞	∞	∞	∞*5.0e4*	0/15
f_{20}	16	851	38111	51362	54470	54861	55313	14/15
	1.6(1)	823(896)	∞	∞	∞	∞	∞*5.0e4*	0/15
f_{21}	41	1157	1674	1692	1705	1729	1757	14/15
	89(306)	119(86)	120(112)	118(111)	117(103)	116(108)	114(157)	3/15
f_{22}	71	386	938	980	1008	1040	1068	14/15
	51(0.8)	259(259)	147(80)	141(306)	137(149)	133(229)	130(175)	4/15
f_{23}	3.0	518	14249	27890	31654	33030	34256	15/15
	2.3(2)	69(145)	∞	∞	∞	∞	∞*5.0e4*	0/15
f_{24}	1622	2.2e5	6.4e6	9.6e6	9.6e6	1.3e7	1.3e7	3/15
	86(154)	∞	∞	∞	∞	∞	∞*5.0e4*	0/15

Table 1: 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 $\downarrow$ symbol, with Bonferroni correction by the number of functions.***
Data produced with COCO v0.0.0

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
	6.6(3)	14(1)	21(1)	29(4)	36(5)	51(4)	66(3)	15/15
f_2	385	386	387	388	390	391	393	15/15
	23(6)	29(8)	32(6)	33(4)	35(5)	37(7)	39(8)	15/15
f_3	5066	7626	7635	7637	7643	7646	7651	15/15
	∞	∞	∞	∞	∞	∞	∞*2.0e5*	0/15
f_4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
	∞	∞	∞	∞	∞	∞	∞*2.0e5*	0/15
f_5	41	41	41	41	41	41	41	15/15
	5.8(2)	6.6(2)	6.7(2)	6.7(2)	6.7(2)	6.7(2)	6.7(2)	15/15
f_6	1296	2343	3413	4255	5220	6728	8409	15/15
	1.7(0.5)	1.5(0.3)	1.4(0.2)	1.4(0.3)	1.5(0.3)	1.6(0.3)	1.7(0.3)	15/15
f_7	1351	4274	9503	16523	16524	16524	16969	15/15
	963(740)	∞	∞	∞	∞	∞	∞*2.0e5*	0/15
f_8	2039	3871	4040	4148	4219	4371	4484	15/15
	2.9(0.5)	16(26)	16(13)	15(36)	15(24)	15(23)	15(34)	12/15
f_9	1716	3102	3277	3379	3455	3594	3727	15/15
	3.8(0.6)	14(0.5)	14(31)	14(0.4)	14(15)	13(28)	13(0.6)	13/15
f_{10}	7413	8661	10735	13641	14920	17073	17476	15/15
	14(7)	27(17)	65(30)	214(513)	198(144)	∞	∞*2.0e5*	0/15
f_{11}	1002	2228	6278	8586	9762	12285	14831	15/15
	429(534)	1342(1010)	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{12}	1042	1938	2740	3156	4140	12407	13827	15/15
	2.9(2)	3.9(3)	3.9(2)	4.0(1)	3.5(2)	1.5(0.7)	1.5(0.3)	15/15
f_{13}	652	2021	2751	3507	18749	24455	30201	15/15
	271(154)	273(223)	1019(2217)	∞	∞	∞	∞*2.0e5*	0/15
f_{14}	75	239	304	451	932	1648	15661	15/15
	2.1(1)	2.4(0.5)	3.4(0.4)	3.9(0.4)	4.2(0.8)	28(23)	∞*2.0e5*	0/15
f_{15}	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
	∞	∞	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{16}	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
	23(37)	∞	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{17}	63	1030	4005	12242	30677	56288	80472	15/15
	1.2(2)	778(776)	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{18}	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
	25(1)	∞	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{19}	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
	1	1	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{20}	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
	3.1(0.7)	∞	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{21}	561	6541	14103	14318	14643	15567	17589	15/15
	55(0.2)	46(84)	28(43)	28(77)	27(17)	26(19)	23(23)	5/15
f_{22}	467	5580	23491	24163	24948	26847	1.3e5	12/15
	286(536)	233(170)	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{23}	3.0	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
	2.3(3)	194(373)	∞	∞	∞	∞	∞*2.0e5*	0/15
f_{24}	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
	∞	∞	∞	∞	∞	∞	∞*2.0e5*	0/15

Table 2: 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 $\downarrow$ symbol, with Bonferroni correction by the number of functions.***
Data produced with COCO v0.0.0

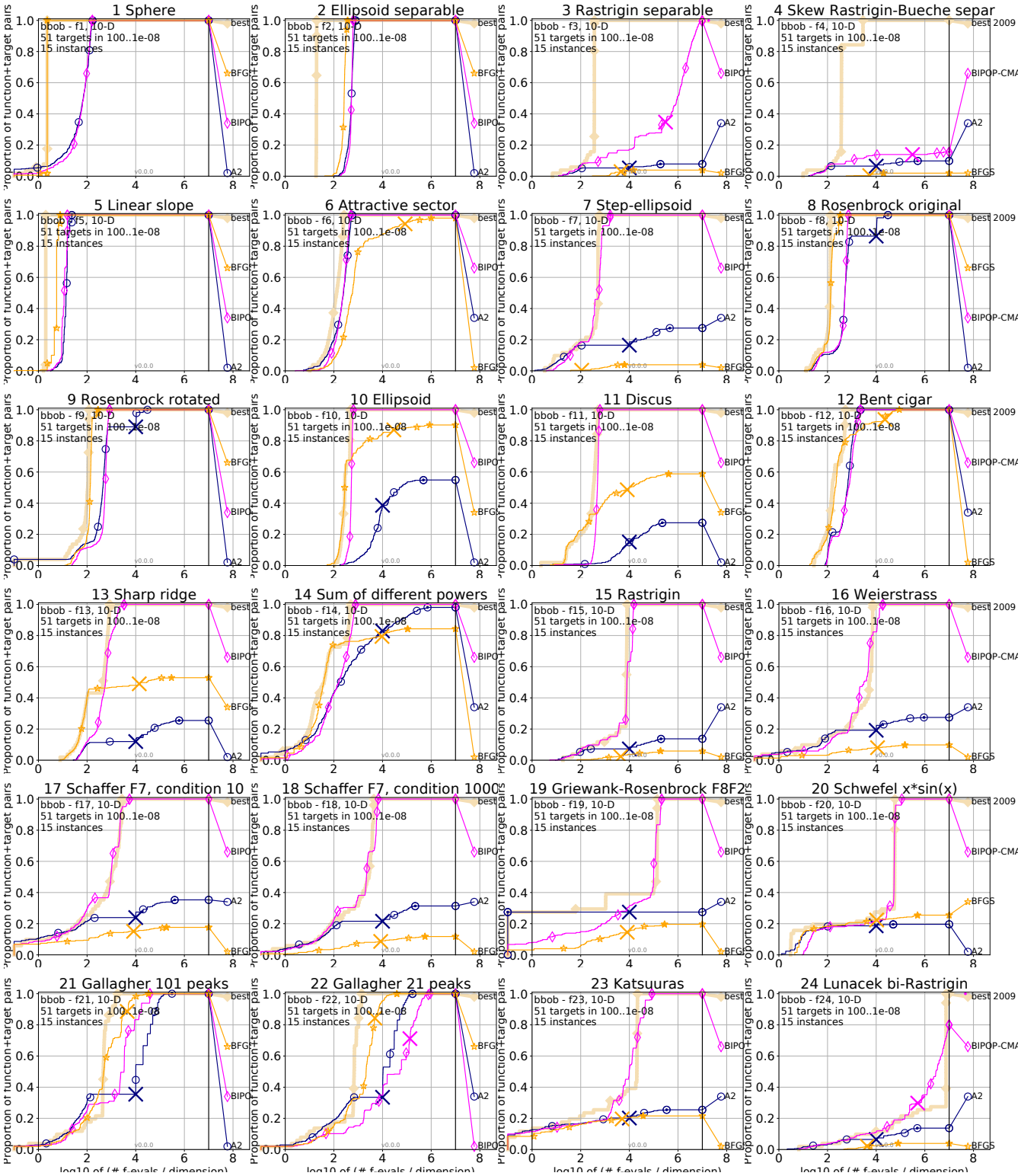

Figure 3: ECDF graphs of AII, BFGS, BIPOP-CMA-ES and the best algorithm from BBOB 2009 for 10D functions

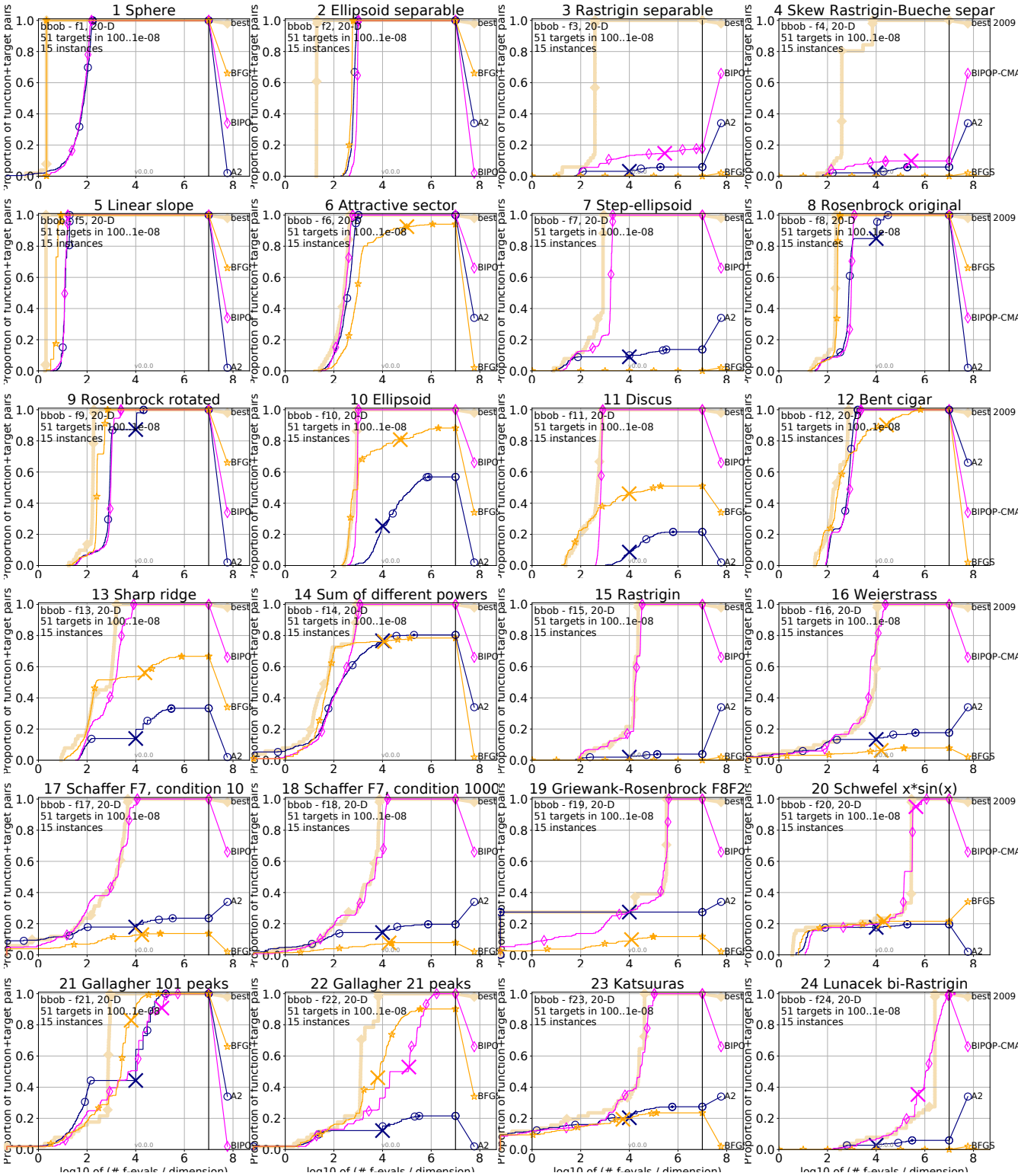

Figure 4: ECDF graphs of AII, BFGS, BIPOP-CMA-ES and the best algorithm from BBOB 2009 for 20D functions

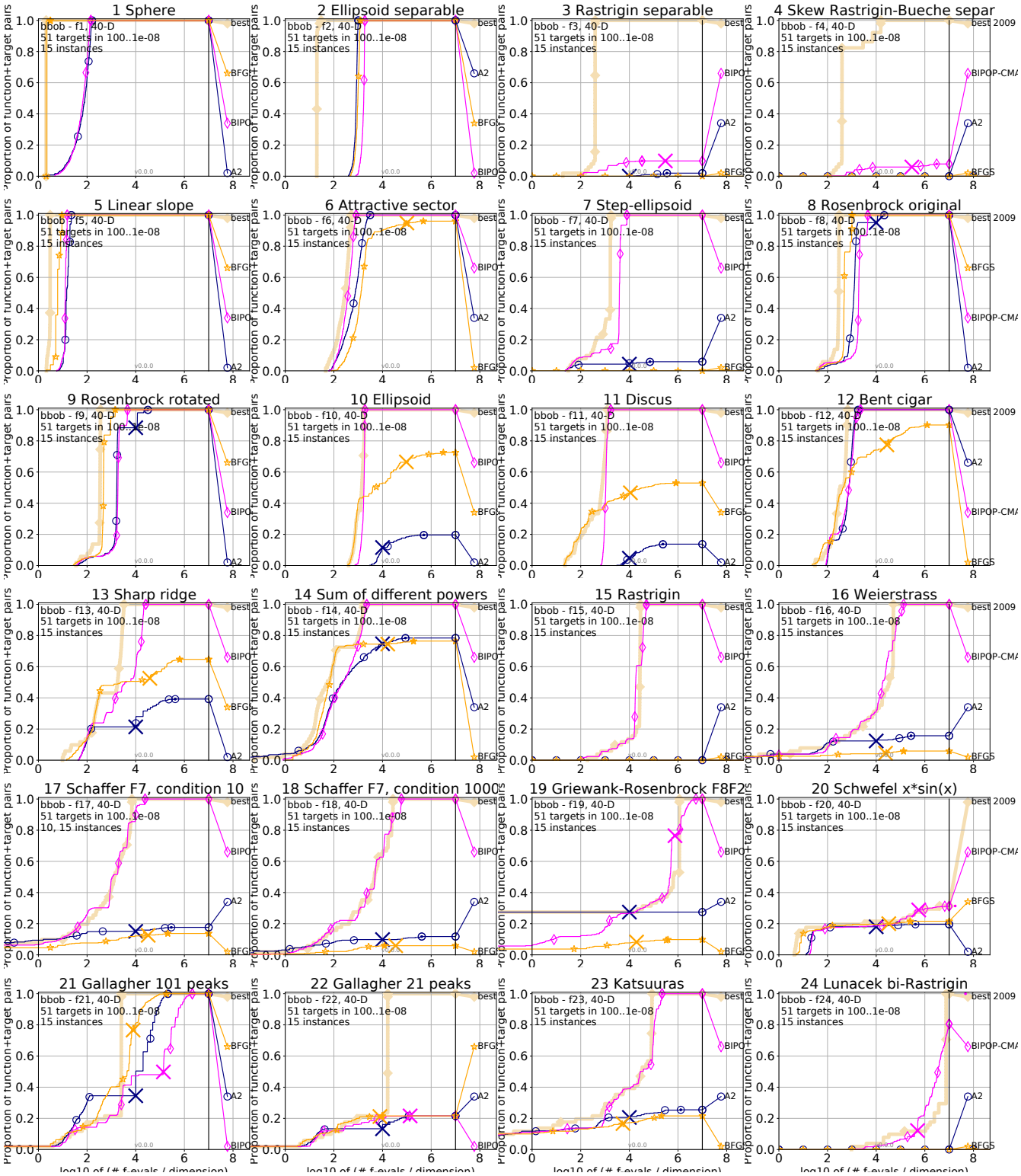

Figure 5: ECDF graphs of AII, BFGS, BIPOP-CMA-ES and the best algorithm from BBOB 2009 for 40D functions