Comparison of Cauchy EDA and pPOEMS algorithms on the BBOB Noiseless Testbed

Petr Pošík
Czech Technical University in Prague
Faculty of Electrical Eng., Dept. of Cybernetics
Technická 2, 166 27 Prague 6, Czech Republic
posik@labe.felk.cvut.cz

Jiří Kubalík
Czech Technical University in Prague
Faculty of Electrical Eng., Dept. of Cybernetics
Technická 2, 166 27 Prague 6, Czech Republic
kubalik@labe.felk.cvut.cz

ABSTRACT

Estimation-of-distribution algorithm using Cauchy sampling distribution is compared with the iterative prototype optimization algorithm with evolved improvement steps. While Cauchy EDA is better on unimodal functions, iterative prototype optimization is more suitable for multimodal functions. This paper compares the results for both algorithms in more detail and adds to the understanding of their key features and differences.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*global optimization, unconstrained optimization*; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Benchmarking, Black-box optimization, Estimation-of-distribution algorithm, Cauchy distribution, POEMS, Iterative prototype optimization with evolved improvement steps

1. INTRODUCTION

The black-box optimization benchmarking (BBOB) workshop in 2009 introduced well-prepared set of benchmark functions suitable for a systematic comparison of black-box optimization algorithms. As an important part of the workshop framework, the whole comparison methodology was created. The 2010 issue of the BBOB methodology [2] allows for a detailed comparison of 2 algorithms on the BBOB functions testbed.

The two algorithms selected for the comparison in this article are:

- The estimation-of-distribution algorithm (EDA) with Cauchy sampling distribution (Cauchy EDA) [7]. The data for this algorithm are taken from the 2009 benchmarking.
- The iterative prototype optimization with evolved improvement steps (POEMS) [4], particularly, one of its variants: POEMS with a pool of candidate prototypes (pPOEMS) described in Sec. 2.1. The data for this algorithm were generated using the 2010 framework.[1]

The comparison is made using the new BBOB 2010 post-processing scripts and templates. Both algorithms fall into the class of evolutionary optimization algorithms, yet they perform in some sense a kind of local search. Their underlying principles are, however, different and it is valueable to look for the effect of their similarities and differences.

In the next section, both algorithms are shortly described and their differences are emphasized. Sec. 3 contains all the results used to compare the algorithms and their discussions. After the presentation of the time demands of both algorithms in Sec. 4, Sec. 5 concludes the paper.

2. ALGORITHM PRESENTATION

The exact description of the Cauchy EDA algorithm along with the parameter settings can be found in [7]. The pPOEMS algorithm is described in the next section.

2.1 pPOEMS

Prototype Optimization with Evolved Improvement Steps (POEMS) optimization algorithm is a stochastic local search algorithm that uses an evolutionary algorithm for searching the neighborhood of the current best solution. The moves in the search space can be thought of as so-called *evolved hypermutations*. The concept of the evolved hypermutations has been shown to outperform other mutation-based evolutionary algorithms that use pure random hypermutations for generating new points in the search space on several combinatorial optimization problems [5, 6].

A description of the original version of the POEMS algorithm for real-valued optimization can be found in [4]. The inner EA evolves hypermutations which are composed

[1]Despite the fact that the algorithms used different versions of the BBOB framework, the results are still comparable. The set of benchmark functions is the same, the only difference is that in 2009 the algorithms were run on 5 instances of each function with 3 repetitions, while in 2010 the algorithms were tested using 15 different instances of the functions.

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

of simple actions of adding a normally distributed real number to each of the coordinates of the prototypes (the length of the action sequence is the same as the search-space dimensionality). The distribution of each single modification is $N(0, \sigma_i^2)$, i.e. the expected step length in each direction is different.

The variant, denoted as pPOEMS, uses a pool of candidate prototypes of size $PoolSize$ from which one prototype is chosen in each iteration. Each candidate prototype maintains its own σ_i^2 values. Thus, the size of the neighborhood to be searched is different for each candidate prototype. The best modification of the current prototype is sought by an evolutionary algorithm and the resulting solution replaces one of the candidate prototypes in the pool according to the following rules:

1. If the modified prototype is better than the current prototype then the modified prototype replaces the current prototype in the pool of prototypes (option A in 1). The values of σ_i^2 of the current prototype are adapted as weighted average of their old values and of the differences between the modified and the current prototype variable values. Finally, this prototype remains the current prototype for the next iteration.

2. If the modified prototype is equally good as the current prototype then it replaces the current prototype in the pool of prototypes (option B in 1) and the values of σ_i^2 of the current prototype are adapted in the same way as in the rule nb. 1. However, since no improvement to the current prototype has been achieved in this iteration the next prototype from the prototype pool (meaning the prototype with index $(i+1)\%PoolSize$, where i is the index of the current prototype) becomes the current prototype for the next iteration.

3. If the modified prototype is worse than the current prototype then the most similar (according to the Euclidean distance) candidate prototype out of the prototypes that have worse fitness than the modified prototype is sought in the pool. If such a prototype exists then it is replaced (option C in 1) by the modified prototype. The values of σ_i^2 of the replacement prototype are adapted according to differences between the modified and the replacement prototype variable values.

   If such a replacement does not exist then the modified prototype is thrown away and the values of σ_i^2 of the current prototype are shrinked as if the differences between the modified and the replacement prototypes were zero. Thus, the values of σ_i^2 are maximally decreased.

   In both cases next prototype from the pool of prototypes becomes the current prototype for the next iteration.

In all cases 1–3, if for some candidate prototype all its σ_i^2 values drop below 10^{-11} then they are reinitialized to $0.25 * (ubound - lbound)$. The prototype itself remains unchanged.

2.2 Algorithm Differences

As already stated, we can look at both algorithms as on quite different instances of local search. The algorithms differ foremost in the following aspects:

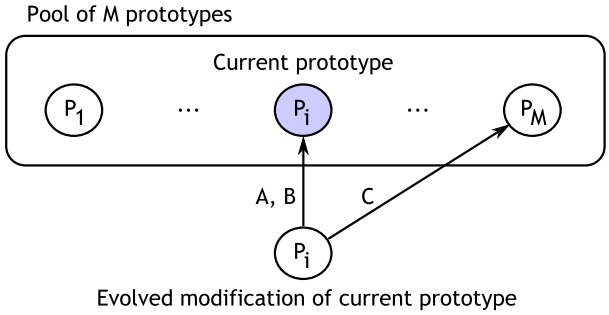

Figure 1: Schema of the pool of prototypes used in pPOEMS.

- In Cauchy EDA, the neighborhood is given by the shape of the Cauchy distribution, which is used to sample new candidate solutions. In pPOEMS, the search is conducted in the neighborhood of the prototype; the neighborhood is given implicitely by the sequence of actions that are evolved to modify the prototype.

- The Cauchy EDA uses population of tens or hundreds of individuals; after its evaluation, the best solution found so far is updated (and the Cauchy-distribution parameters as well). In pPOEMS, the prototype is updated after certain number of generations spent in the inner EA which evolves the improving sequence of actions. The inner EA run is longer (it runs for $200 \times D$ fitness evaluations) and the updates of prototypes are therefore less frequent, especially for higher dimensions.

- Both algorithms can get stuck in local optima. The Cauchy EDA prevents this situation by independent restarts, as suggested in the BBOB methodology. The pPOEMS algorithm is not restarted; instead it maintains a pool of candidate prototypes which can be used in situations when the inner EA is not able to evolve an improving sequence from the current prototype.

For both algorithms, the crafting effort CrE= 0.

3. RESULTS

Results from experiments according to [2] on the benchmark functions given in [1, 3] are presented in Figures 2, 3 and 4 and in Table 1. The **expected running time (ERT)**, used in the figures and table, 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 [2, 8]. **Statistical significance** is tested with the rank-sum test for a given target Δf_t (10^{-8} in Figure 2) 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.

Cauchy EDA outperforms the pPOEMS algorithm on functions 1, 2, 5, 7, 9, 10, 11, 12, 13, 14, 17, 18 for the tight target

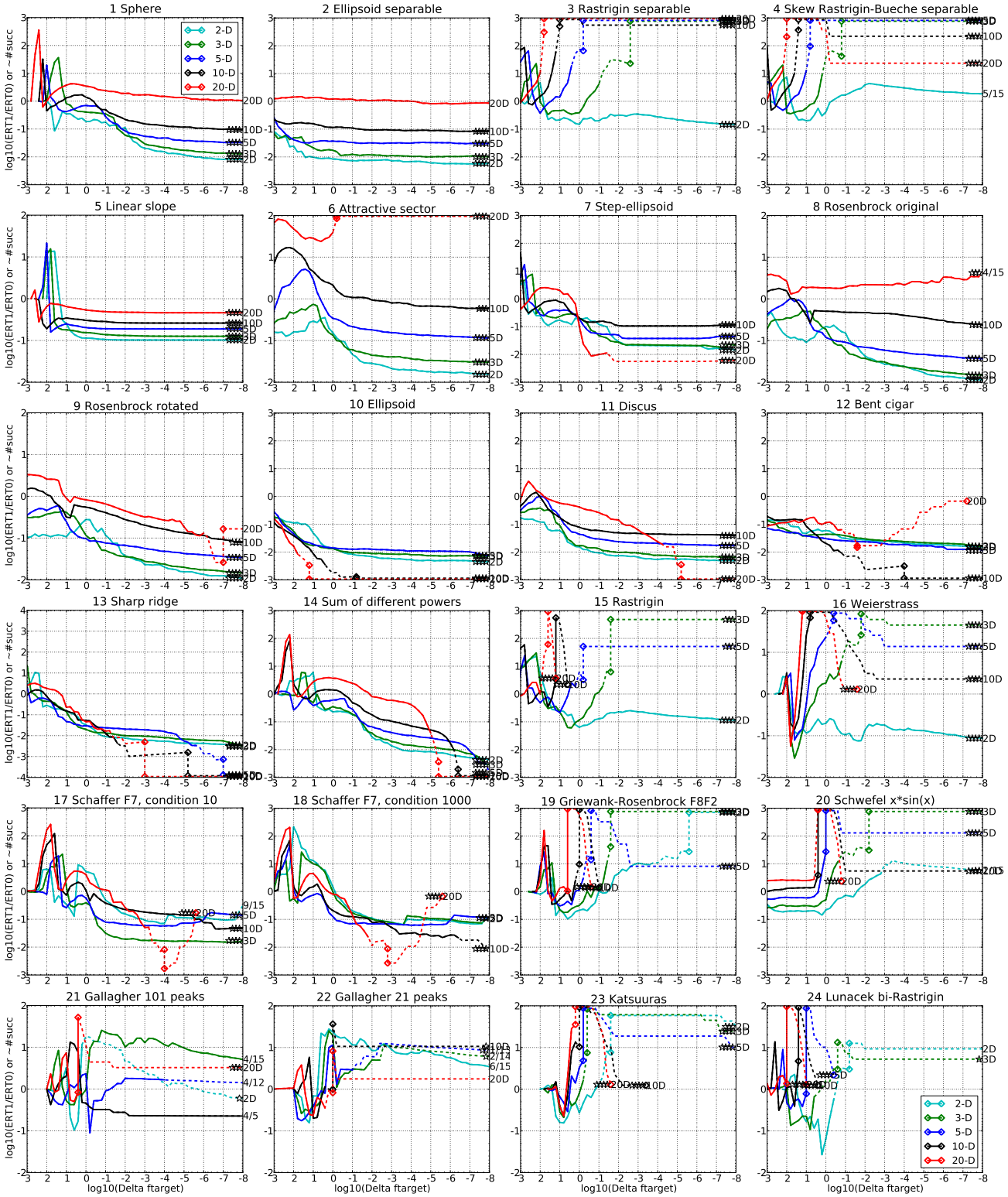

Figure 2: ERT ratio of CauchyEDA divided by pPOEMS versus $\log_{10}(\Delta f)$ for f_1–f_{24} in 2, 3, 5, 10, 20, 40-D. Ratios $< 10^0$ indicate an advantage of CauchyEDA, smaller values are always better. The line gets dashed when for any algorithm the ERT exceeds thrice the median of the trial-wise overall number of f-evaluations for the same algorithm on this function. Symbols indicate the best achieved Δf-value of one algorithm (ERT gets undefined to the right). The dashed line continues as the fraction of successful trials of the other algorithm, where 0 means 0% and the y-axis limits mean 100%, values below zero for CauchyEDA. The line ends when no algorithm reaches Δf anymore. The number of successful trials is given, only if it was in $\{1 \dots 9\}$ for CauchyEDA (1st number) and non-zero for pPOEMS (2nd number). Results are significant with $p = 0.05$ for one star and $p = 10^{-\#\star}$ otherwise, with Bonferroni correction within each figure.

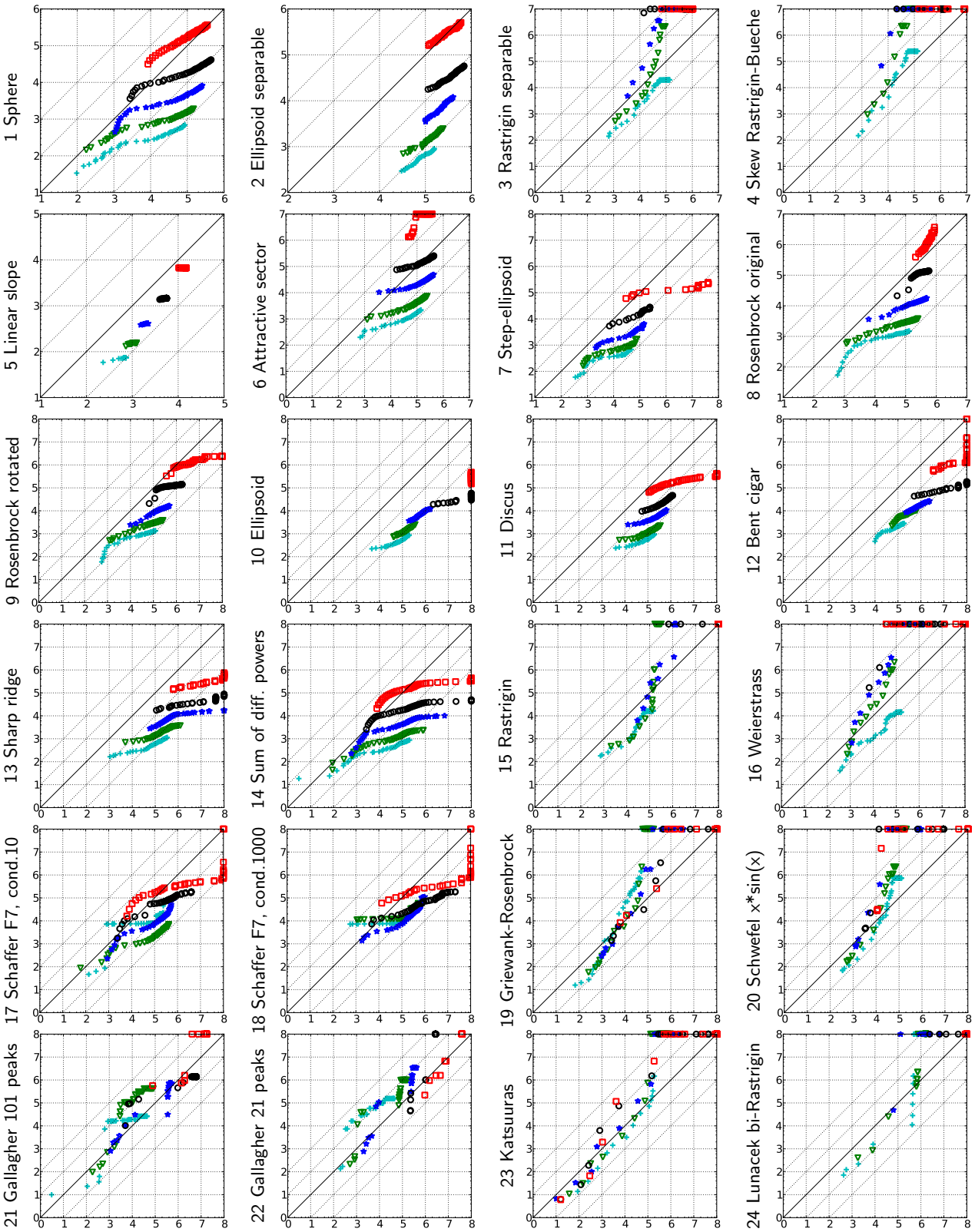

Figure 3: Expected running time (ERT in log10 of number of function evaluations) of CauchyEDA versus pPOEMS for 46 target values $\Delta f \in [10^{-8}, 10]$ in each dimension for functions f_1–f_{24}. Markers on the upper or right egde indicate that the target value was never reached by CauchyEDA or pPOEMS respectively. Markers represent dimension: 2:+, 3:▽, 5:⋆, 10:○, 20:□, 40:◇.

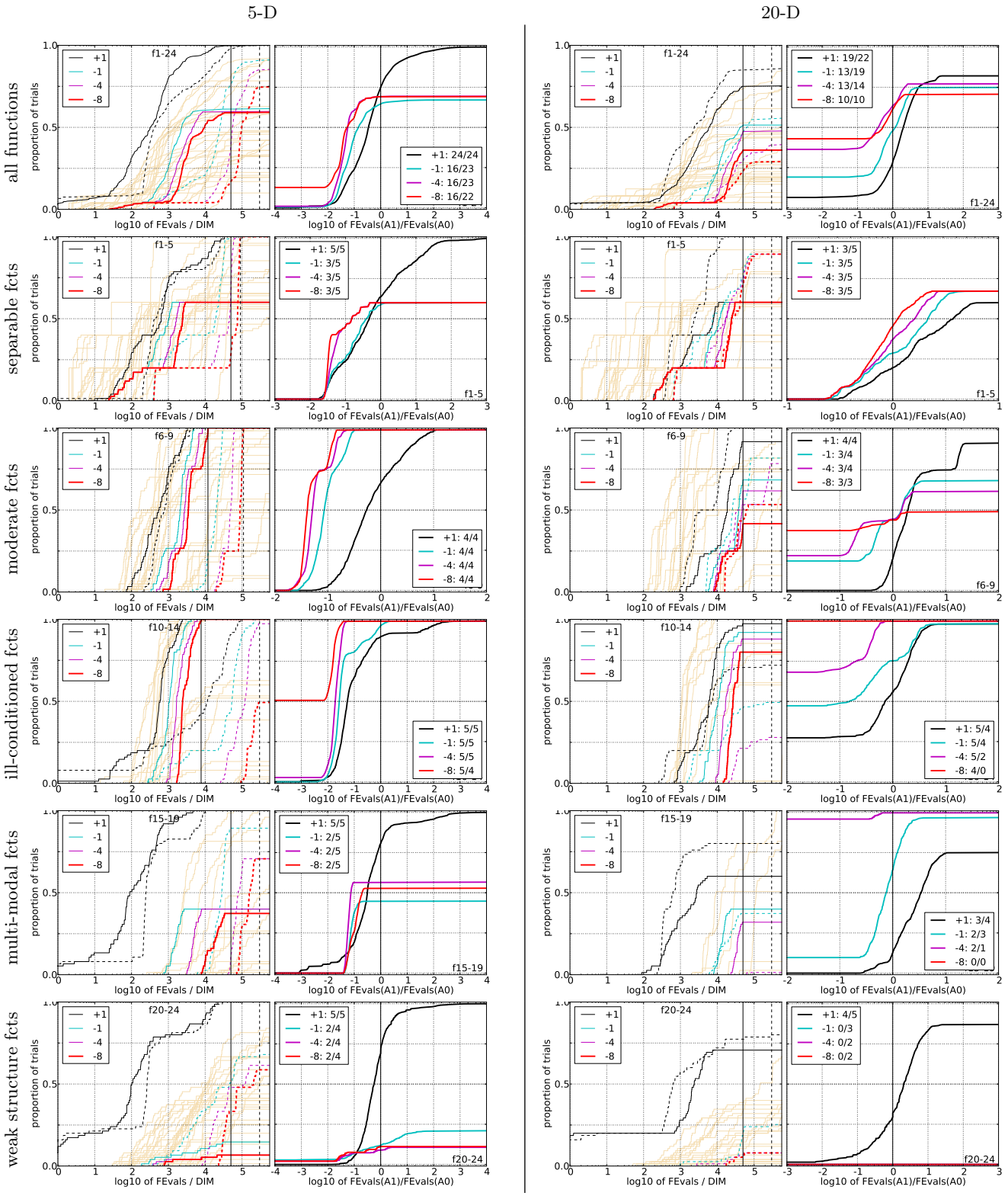

Figure 4: Empirical cumulative distributions (ECDF) of run lengths and speed-up ratios in 5-D (left) and 20-D (right). Left sub-columns: ECDF of the number of function evaluations divided by dimension D (FEvals/D) to reach a target value $f_{\text{opt}} + \Delta f$ with $\Delta f = 10^k$, where $k \in \{1, -1, -4, -8\}$ is given by the first value in the legend, for CauchyEDA (solid) and pPOEMS (dashed). Light beige lines show the ECDF of FEvals for target value $\Delta f = 10^{-8}$ of algorithms benchmarked during BBOB-2009. Right sub-columns: ECDF of FEval ratios of CauchyEDA divided by pPOEMS, all trial pairs for each function. Pairs where both trials failed are disregarded, pairs where one trial failed are visible in the limits being > 0 or < 1. The legends indicate the number of functions that were solved in at least one trial (CauchyEDA first).

5-D

Δf	1e+1	1e+0	1e-1	1e-3	1e-5	1e-7	#succ
$\mathbf{f_1}$	11	12	12	12	12	12	15/15
0: pPO	97	130	610	5.8e3	1.2e4	1.8e4	15/15
1: Cau	**41**$^{\star 2}$	90	**170**$^{\star 2}$	**310**$^{\star 3}$	**460**$^{\star 3}$	**600**$^{\star 3}$	15/15
$\mathbf{f_2}$	83	87	88	90	92	94	15/15
0: pPO	1.2e3	1.3e3	1.7e3	2.5e3	3.1e3	3.9e3	15/15
1: Cau	**42**$^{\star 3}$	**49**$^{\star 3}$	**58**$^{\star 3}$	**80**$^{\star 3}$	**100**$^{\star 3}$	**120**$^{\star 3}$	15/15
$\mathbf{f_3}$	720	1600	1600	1600	1700	1700	15/15
0: pPO	4.8	**30**$^{\star 3}$	**64**$^{\star 3}$	**120**$^{\star 3}$	**160**$^{\star 3}$	**200**$^{\star 3}$	15/15
1: Cau	6.7	2.2e3	∞	∞	∞	∞*2.5e5*	0/15
$\mathbf{f_4}$	810	1600	1700	1800	1900	1900	15/15
0: pPO	**6.8**$^{\star 3}$	**36**$^{\star 3}$	**79**$^{\star 3}$	**120**$^{\star 3}$	**160**$^{\star 3}$	**200**$^{\star 3}$	15/15
1: Cau	85	∞	∞	∞	∞	∞*2.5e5*	0/15
$\mathbf{f_5}$	10	10	10	10	10	10	15/15
0: pPO	150	200	210	220	220	220	15/15
1: Cau	**39**$^{\star 3}$	**41**$^{\star 3}$	**41**$^{\star 3}$	**41**$^{\star 3}$	**41**$^{\star 3}$	**41**$^{\star 3}$	15/15
$\mathbf{f_6}$	110	210	280	580	1000	1300	15/15
0: pPO	**31**$^{\star 2}$	200	310	310	270	280	15/15
1: Cau	92	**69**$^{\star 3}$	**68**$^{\star 3}$	**47**$^{\star 3}$	**35**$^{\star 3}$	**34**$^{\star 3}$	15/15
$\mathbf{f_7}$	24	320	1200	1600	1600	1600	15/15
0: pPO	89	26	43	76	76	79	15/15
1: Cau	**33**$^{\star 3}$	**4.9**$^{\star 3}$	**2.4**$^{\star 3}$	**2.9**$^{\star 3}$	**2.9**$^{\star 3}$	**3.4**$^{\star 3}$	15/15
$\mathbf{f_8}$	73	270	340	390	410	420	15/15
0: pPO	84	260	370	580	800	1.1e3	15/15
1: Cau	49	**31**$^{\star 3}$	**33**$^{\star 3}$	**34**$^{\star 3}$	**37**$^{\star 3}$	**40**$^{\star 3}$	15/15
$\mathbf{f_9}$	35	130	210	300	340	370	15/15
0: pPO	290	560	600	720	990	1.2e3	15/15
1: Cau	**71**$^{\star 2}$	**54**$^{\star 3}$	**45**$^{\star 3}$	**41**$^{\star 3}$	**42**$^{\star 3}$	**43**$^{\star 3}$	15/15
$\mathbf{f_{10}}$	350	500	570	630	830	880	15/15
0: pPO	550	640	740	1.0e3	1.0e3	1.2e3	11/15
1: Cau	**11**$^{\star 3}$	**9**$^{\star 3}$	**9.4**$^{\star 3}$	**12**$^{\star 3}$	**11**$^{\star 3}$	**13**$^{\star 3}$	15/15
$\mathbf{f_{11}}$	140	200	760	1200	1500	1700	15/15
0: pPO	95	390	200	250	290	340	15/15
1: Cau	**18**$^{\star 2}$	**17**$^{\star 3}$	**6**$^{\star 3}$	**5.3**$^{\star 3}$	**5.6**$^{\star 3}$	**5.9**$^{\star 3}$	15/15
$\mathbf{f_{12}}$	110	270	370	460	1300	1500	15/15
0: pPO	2.1e3	1.4e3	1.4e3	1.9e3	1.0e3	1.4e3	8/15
1: Cau	**79**$^{\star 3}$	**41**$^{\star 3}$	**35**$^{\star 3}$	**38**$^{\star 3}$	**17**$^{\star 3}$	**17**$^{\star 3}$	15/15
$\mathbf{f_{13}}$	130	190	250	1300	1800	2300	15/15
0: pPO	450	790	1.1e3	390	1.3e3	1.0e4	0/15
1: Cau	**21**$^{\star 3}$	**24**$^{\star 3}$	**25**$^{\star 3}$	**7.4**$^{\star 3}$	**7.3**$^{\star 3}$	**7.3**$^{\star 3}$	15/15
$\mathbf{f_{14}}$	9.8	41	58	140	250	480	15/15
0: pPO	61	47	140	800	1.1e3	2.1e3	3/15
1: Cau	23	**29***	**40***	**33**$^{\star 3}$	**28**$^{\star 3}$	**19**$^{\star 3}$	15/15
$\mathbf{f_{15}}$	510	9300	1.9e4	2.0e4	2.1e4	2.1e4	14/15
0: pPO	57	**29***	**60**$^{\star 2}$	**62**$^{\star 2}$	**64**$^{\star 2}$	**66**$^{\star 2}$	9/15
1: Cau	**12**$^{\star 2}$	190	∞	∞	∞	∞*2.5e5*	0/15
$\mathbf{f_{16}}$	120	610	2700	1.0e4	1.2e4	1.2e4	15/15
0: pPO	9.6	**52***	**40**$^{\star 3}$	**120**$^{\star 3}$	**110**$^{\star 3}$	**120**$^{\star 3}$	9/15
1: Cau	5.6	1.2e3	∞	∞	∞	∞*2.5e5*	0/15
$\mathbf{f_{17}}$	5.2	210	900	3700	6400	7900	15/15
0: pPO	170	22	88	71	73	82	15/15
1: Cau	44	13	**7**$^{\star 3}$	**4.3**$^{\star 3}$	**5.3**$^{\star 3}$	**13**$^{\star 3}$	14/15
$\mathbf{f_{18}}$	100	380	4000	9300	1.1e4	1.2e4	15/15
0: pPO	18	95	34	39	55	69	15/15
1: Cau	13	**12**$^{\star 3}$	**2.4**$^{\star 3}$	**2.7**$^{\star 3}$	**3.7**$^{\star 3}$	**8.6**$^{\star 3}$	14/15
$\mathbf{f_{19}}$	1	1	240	1.2e5	1.2e5	1.2e5	15/15
0: pPO	980	1.8e4	**1.6e3**$^{\star 3}$	**27**$^{\star 3}$	**27**$^{\star 3}$	**27**$^{\star 3}$	5/15
1: Cau	**300***	2.1e4	∞	∞	∞	∞*2.5e5*	0/15
$\mathbf{f_{20}}$	16	850	3.8e4	5.4e4	5.5e4	5.5e4	14/15
0: pPO	81	**17**$^{\star 2}$	**17**$^{\star 3}$	**13**$^{\star 3}$	**14**$^{\star 3}$	**15**$^{\star 3}$	11/15
1: Cau	**48***	460	∞	∞	∞	∞*2.5e5*	0/15
$\mathbf{f_{21}}$	41	1200	1700	1700	1700	1800	14/15
0: pPO	28	11	210	240	260	280	12/15
1: Cau	20	27	190	420	420	410	4/15
$\mathbf{f_{22}}$	71	390	940	1000	1000	1100	14/15
0: pPO	29	24	270	280	330	360	13/15
1: Cau	**11***	280	780	3.5e3	3.4e3	3.3e3	1/15
$\mathbf{f_{23}}$	3	520	1.4e4	3.2e4	3.3e4	3.4e4	15/15
0: pPO	3.4	71	**29**$^{\star 3}$	**42**$^{\star 3}$	**47**$^{\star 3}$	**53**$^{\star 3}$	8/15
1: Cau	2.2	230	∞	∞	∞	∞*2.5e5*	0/15
$\mathbf{f_{24}}$	1600	2.2e5	6.4e6	9.6e6	1.3e7	1.3e7	3/15
0: pPO	39	**7.2**$^{\star 3}$	∞	∞	∞	∞*1.5e6*	0/15
1: Cau	30	∞	∞	∞	∞	∞*2.5e5*	0/15

20-D

Δf	1e+1	1e+0	1e-1	1e-3	1e-5	1e-7	#succ
$\mathbf{f_1}$	43	43	43	43	43	43	15/15
0: pPO	**200**$^{\star 3}$	**470**$^{\star 3}$	**1.1e3**$^{\star 3}$	**2.6e3***	4.5e3	7.1e3	15/15
1: Cau	730	1.6e3	2.5e3	4.3e3	6.1e3	7.8e3	15/15
$\mathbf{f_2}$	380	390	390	390	390	390	15/15
0: pPO	**310**$^{\star 2}$	420	510	740	1.2e3	1.4e3	15/15
1: Cau	410	510	610	800	990	1.2e3	15/15
$\mathbf{f_3}$	5100	7600	7600	7600	7600	7700	15/15
0: pPO	**13**$^{\star 3}$	**57**$^{\star 3}$	**100**$^{\star 3}$	**110**$^{\star 3}$	**120**$^{\star 3}$	**130**$^{\star 3}$	15/15
1: Cau	∞	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_4}$	4700	7600	7700	7700	7800	1.4e5	9/15
0: pPO	**31**$^{\star 3}$	**250**$^{\star 3}$	**1.0e3**$^{\star 3}$	**1.1e3**$^{\star 3}$	**1.1e3**$^{\star 3}$	**59**$^{\star 3}$	7/15
1: Cau	∞	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_5}$	41	41	41	41	41	41	15/15
0: pPO	250	310	340	350	360	360	15/15
1: Cau	**160**$^{\star 2}$	**170**$^{\star 3}$	**170**$^{\star 3}$	**170**$^{\star 3}$	**170**$^{\star 3}$	**170**$^{\star 3}$	15/15
$\mathbf{f_6}$	1300	2300	3400	5200	6700	8400	15/15
0: pPO	**36**$^{\star 3}$	**32**$^{\star 3}$	**32**$^{\star 3}$	**34**$^{\star 3}$	**35**$^{\star 3}$	**38**$^{\star 3}$	15/15
1: Cau	1.0e3	1.3e3	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_7}$	1400	4300	9500	1.7e4	1.7e4	1.7e4	15/15
0: pPO	**22**$^{\star 2}$	280	1.8e3	2.4e3	2.4e3	2.3e3	2/15
1: Cau	44	**29***	**18**$^{\star 2}$	**14**$^{\star 3}$	**14**$^{\star 3}$	**14**$^{\star 3}$	15/15
$\mathbf{f_8}$	2000	3900	4000	4200	4400	4500	15/15
0: pPO	**100**$^{\star 2}$	**97**$^{\star 3}$	**110**$^{\star 3}$	**120**$^{\star 3}$	**150**$^{\star 2}$	190	15/15
1: Cau	190	180	210	260	360	540	4/15
$\mathbf{f_9}$	1700	3100	3300	3500	3600	3700	15/15
0: pPO	210	330	440	970	2.5e3	2.4e4	0/15
1: Cau	190	270	290	310	470	630	6/15
$\mathbf{f_{10}}$	7400	8700	1.1e4	1.5e4	1.7e4	1.7e4	15/15
0: pPO	∞	∞	∞	∞	∞	∞*6.0e6*	0/15
1: Cau	**20**$^{\star 3}$	**22**$^{\star 3}$	**20**$^{\star 3}$	**20**$^{\star 3}$	**21**$^{\star 3}$	**25**$^{\star 3}$	15/15
$\mathbf{f_{11}}$	1000	2200	6300	9800	1.2e4	1.5e4	15/15
0: pPO	110	110	84	360	7.1e3	∞*6.0e6*	0/15
1: Cau	**64**$^{\star 3}$	**44**$^{\star 3}$	**22**$^{\star 3}$	**22**$^{\star 3}$	**25**$^{\star 3}$	**26**$^{\star 3}$	15/15
$\mathbf{f_{12}}$	1000	1900	2700	4100	1.2e4	1.4e4	15/15
0: pPO	3.4e3	3.9e3	9.3e3	∞	∞	∞*6.0e6*	0/15
1: Cau	510	440	420	380	390	1.1e3	0/15
$\mathbf{f_{13}}$	650	2000	2800	1.9e4	2.4e4	3.0e4	15/15
0: pPO	940	2.2e3	6.2e3	4.6e3	∞	∞*6.0e6*	0/15
1: Cau	210	**100***	**100**$^{\star 3}$	**23**$^{\star 3}$	**23**$^{\star 3}$	**23**$^{\star 3}$	15/15
$\mathbf{f_{14}}$	75	240	300	930	1600	1.6e4	15/15
0: pPO	**100**$^{\star 3}$	**72**$^{\star 3}$	**140**$^{\star 3}$	350	4.5e3	∞*6.0e6*	0/15
1: Cau	280	270	350	**210**$^{\star 2}$	**180**$^{\star 3}$	**25**$^{\star 3}$	15/15
$\mathbf{f_{15}}$	3.0e4	1.5e5	3.1e5	3.2e5	4.5e5	4.6e5	15/15
0: pPO	∞	∞	∞	∞	∞	∞*6.0e6*	0/15
1: Cau	∞	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_{16}}$	1400	2.7e4	7.7e4	1.9e5	2.0e5	2.2e5	15/15
0: pPO	**26**$^{\star 3}$	**24**$^{\star 3}$	**1.1e3**$^{\star 3}$	∞	∞	∞*6.0e6*	0/15
1: Cau	∞	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_{17}}$	63	1000	4000	3.1e4	5.6e4	8.0e4	15/15
0: pPO	98	**31**$^{\star 3}$	54	310	∞	∞*6.0e6*	0/15
1: Cau	260	120	62	**16***	**23**$^{\star 3}$	∞*1.0e6*	0/15
$\mathbf{f_{18}}$	620	4000	2.0e4	6.8e4	1.3e5	1.5e5	15/15
0: pPO	**21**$^{\star 3}$	51	110	∞	∞	∞*6.0e6*	0/15
1: Cau	96	42	**15**$^{\star 3}$	**12**$^{\star 3}$	**38**$^{\star 3}$	∞*1.0e6*	0/15
$\mathbf{f_{19}}$	1	1	3.4e5	6.2e6	6.7e6	6.7e6	15/15
0: pPO	6.3e3	**1.4e6**$^{\star 3}$	∞	∞	∞	∞*6.0e6*	0/15
1: Cau	8.4e3	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_{20}}$	82	4.6e4	3.1e6	5.5e6	5.6e6	5.6e6	14/15
0: pPO	**130**$^{\star 3}$	**1.8**$^{\star 3}$	∞	∞	∞	∞*6.0e6*	0/15
1: Cau	340	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_{21}}$	560	6500	1.4e4	1.5e4	1.6e4	1.8e4	15/15
0: pPO	140	1.2e3	960	1.2e3	1.1e3	970	4/15
1: Cau	1.0e3	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_{22}}$	470	5600	2.3e4	2.5e4	2.7e4	1.3e5	12/15
0: pPO	2.0e3	1.4e3	1.7e3	1.6e3	1.5e3	290	2/15
1: Cau	470	1.2e3	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_{23}}$	3.2	1600	6.7e4	4.9e5	8.1e5	8.4e5	15/15
0: pPO	4.9	**320**$^{\star 3}$	**36**$^{\star 3}$	∞	∞	∞*6.0e6*	0/15
1: Cau	1.9	∞	∞	∞	∞	∞*1.0e6*	0/15
$\mathbf{f_{24}}$	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	3/15
0: pPO	**64**$^{\star 3}$	∞	∞	∞	∞	∞*6.0e6*	0/15
1: Cau	∞	∞	∞	∞	∞	∞*1.0e6*	0/15

Table 1: Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 (given in the respective first row) for different Δf values for functions f_1–f_{24}. The median number of conducted function evaluations is additionally given in *italics*, if $\mathrm{ERT}(10^{-7}) = \infty$. #succ is the number of trials that reached the final target $f_{\mathrm{opt}} + 10^{-8}$. 0: pPO is pPOEMS and 1: Cau is CauchyEDA. Bold entries are statistically significantly better compared to the other algorithm, with $p = 0.05$ or $p = 10^{-k}$ where $k > 1$ is the number following the $\star$ symbol, with Bonferroni correction of 48.

Table 2: The average time demands per function evaluation (in microseconds) of the two compared algorithms.

Dim	2	3	5	10	20
pPOEMS	2.4	2.7	4	8.5	13
CauchyEDA	51	17	9	9	11

values, while pPOEMS beats Cauchy EDA on functions 3, 4, 15, 16, 19, 20, 22. (The results on the other functions are mixed, or neither algorithm solved the problem successfully.) In this small competition the Cauchy EDA wins 12:7. It can be stated that Cauchy EDA beats pPOEMS mainly on the unimodal functions, while pPOEMS is better on multimodal ones.

The pPOEMS algorithm is not invariant with respect to the rotation of the search space. The evolved actions operate on 1 dimension only (making axis-parallel modifications only), nevertheless, the whole action sequences (hypermutations) result in non-axis-paralel steps. However, it is easier for this algorithm to optimize separable functions. As an example, we can look at function pair 2-10 in Fig. 2: the non-rotated version of the ellipsoid function is much easier for pPOEMS, while it cannot solve the rotated version in higher dimensions at all. And even though the Rosenbrock function (8) is not separable, its rotated version (9) is for pPOEMS much harder as well.

In lower dimensions, Cauchy EDA is often orders of magnitude faster than pPOEMS. With increasing dimensionality this gap reduces, and the speed of both algorithms becomes almost equal (e.g. for 20D versions of sphere and ellipsoid functions). It may be anticipated that for larger dimensions pPOEMS would overtake the Cauchy EDA.

Looking at Fig. 4, it can be stated that pPOEMS beats CauchyEDA clearly on separable functions. It is better also for multimodal and weak-structure functions, but neither algorithm is really successful on these (especially in 20D). Cauchy EDA is a clear winner for ill-conditioned functions and for moderate functions in lower dimensions.

4. CPU TIMING EXPERIMENTS

The time requirements of Cauchy EDA are taken from [7]. The multistart algorithm was run with the maximal number of evaluations set to 10^5, the basic algorithm was restarted for at least 30 seconds. These experiments have been conducted on Intel Core 2 CPU, T5600, 1.83 GHz, 1 GB RAM with Windows XP SP3 in MATLAB R2007b for Cauchy EDA and on Intel Pentium-M 1400 MHz with Windows XP SP3 using the implementation in C for pPOEMS. The comparison of the average time demands per function evaluation are shown in Table 2.

The differences in the average time needed for function evaluation are caused by the fact that CauchyEDA was implemented in MATLAB while pPOEMS in C. The MATLAB implementation becomes more efficient for larger populations.

5. CONCLUSIONS

Cauchy EDA and POEMS algorithm with a pool of candidate prototypes were compared using the BBOB 2010 methodology. The results confirm that pPOEMS searches much broader neighborhood than Cauchy EDA. The pPOEMS algorithm is able to solve certain percentage of multimodal functions, while the performace of Cauchy EDA is for them rather weak (and the restarting does not help much).

The pPOEMS algorithm is rather slow (compared to Cauchy EDA and other algorithms taking part in BBOB 2009) which showed up especially on unimodal functions. The pPOEMS greatly suffers from the fact that the individual actions in the improving sequence operate over axis-parallel directions. For non-separable functions, this renders the crossover operator used in the inner EA of POEMS rather useless since the individual actions are correlated. To incorporate a method that would decorrelate the actions between the individual inner-EA launches remains as future work.

Acknowledgements

Petr Pošík is supported by the Grant Agency of the Czech Republic with the grant no. 102/08/P094 entitled "Machine learning methods for solution construction in evolutionary algorithms". Jiří Kubalík is supported by the Ministry of Education, Youth and Sport of the Czech Republic with the grant No. MSM6840770012 entitled "Transdisciplinary Research in Biomedical Engineering II".

6. REFERENCES

[1] S. Finck, N. 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. Updated February 2010.

[2] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report RR-7215, INRIA, 2010.

[3] 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. Updated February 2010.

[4] J. Kubalik. Black-box optimization benchmarking of prototype optimization with evolved improvement steps for noiseless function testbed. In F. Rothlauf, editor, *GECCO (Companion)*, pages 2303–2308. ACM, 2009.

[5] J. Kubalík. Solving the sorting network problem using iterative optimization with evolved hypermutations. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 301–308, New York, NY, USA, 2009. ACM.

[6] J. Kubalík. Efficient stochastic local search algorithm for solving the shortest common supersequence problem. In *Accepted for presentation at GECCO '10*. ACM, 2010.

[7] P. Pošík. BBOB-benchmarking a simple estimation-of-distribution algorithm with Cauchy distribution. In *GECCO '09: Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference*, pages 2309–2314, New York, NY, USA, 2009. ACM.

[8] 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.