Optimization-Group Project
$(\mu/\mu, \lambda) - ES$ with Search Path

Final Report *

Answers:
CHEN Xiaoxiao, FEI Dong
LI Honglin, WANG Yuxiang

ABSTRACT

Evolution strategies are evolutationary algorithms inspired by biological evolution, which are applied to black-box optimization problems. This kind of optimization method consistants to sample new candidate solutions stochastically. We have chosen one of the evolutationary algorithms, by using the platform for Comparing Continuous Optimizers in a black-box setting (COCO), to test the performance of our algorithm.

Keywords

Benchmarking, Black-box optimization

1. INTRODUCTION

$(\mu/\mu, \lambda) - ES$, an Evolution Strategy with recombination of all parents during each generation, especially focuses on solving the problem of continous black-box optimization. As one of the covariance matrix adaptation evolution strategies, it is a stochastic, population-based search method in continous search space, aiming at minimizing an single objective function $f : \mathbb{R}^n \to \mathbb{R}, x \to f(x)$. Ameliorated from the three preceding algorithms in the same article, we try to realize this algothm and do the benchmarking of its performances on the coco platform. As the data set mesured by other algorithm on coco is aboundant, we can compare the performance of various algorithm. Each algorithm is mesured under same condition with the same given objective functions, which makes our result much more persuasive. Through the experience implemented on a benchmarking platform, we get familiar with the method to realize a black-box optimization algorithm and gradully adjust the parameters aiming for reaching the best performances.

2. ALGORITHM PRESENTATION

Inspired by biological evolution, the basic formulation of our algorithm is based on the application of mutation, recombination and selection in populations of candidate solution. When the result meets the predefined criterion, we consider achieving the expected solution. $(\mu/\mu, \lambda) - ES$ with Search Path uses self-adaptation step-size with an additional search path control. We assume a population $\mathbb{P}$, of so-called individuals and each individual consists of a solution $x \in \mathbb{R}^n$. In this algorithm, in each iteration step k, λ new generations $x_i \in \mathbb{R}^n, i = 1, ..., \lambda$, are generated by sampling a multi-variate normal distribution, $\mathbb{N}(m, C_k)$, with mean of the parents and $n \times n$ covariance matrix C_k. Then, The μ best solutions are selected to update the distribution parameters for next iteration.

0 given $n \in \mathbb{N}_+, \lambda \in \mathbb{N}, \mu \approx \lambda/4 \in \mathbb{N}, c_\sigma \approx \sqrt{\mu/(n+\mu)}, d \approx 1 + \sqrt{\mu/n}, d_i \approx 3n$
1 initialize $x \in \mathbb{R}^n, \sigma \in \mathbb{R}_+^N, s_\sigma = 0$
2 while not happy
3 for $k \in 1, ..., \lambda$
4 $z_k = \mathbb{N}(0, \mathbb{I})$
5 $x_k = x + \sigma \circ z_k$
6 $\mathbb{P} \leftarrow sel\_\mu\_best((x_k, z_k, f(x_k)) \mid 1 \leqslant k \leqslant \lambda)$
// recombination and parent update
7 $s_\sigma \leftarrow (1 - c_\sigma)s_\sigma + \sqrt{c_\sigma(2 - c_\sigma)}\frac{\sqrt{\mu}}{u} \sum_{z_k \in \mathbb{P}} z_k$
8 $\sigma \leftarrow \sigma \circ exp^{1/d_i}(\frac{|s_\sigma|}{\mathbb{E}|\mathbb{N}(0,1)|} - 1) \times exp^{c_\sigma/d}(\frac{\|s_\sigma\|}{\mathbb{E}\|\mathbb{N}(0,\mathbb{I})\|} - 1)$
9 $x = \frac{1}{\mu} \sum_{x_k \in \mathbb{P}} x_k$

3. EXPERIMENTAL PROCEDURE

3.1 Implementation

Implementation of algorithm

We chose to implement this algorithm with language Python. To verify that the algorithm is working properly, we tried to realize the algorithm first in a simple way and did the unit test. Then under the coco framework, the ES-with-Search-Path algorithm was implemented and tested for different dimensions with different budgets.

Our algorithm consists of three parts as three functions listed below: two stand for the critical steps during the algorithm procedure, another one plays the role of the principal function.

$\boxed{ES\_search\_path(fun, lbounds, ubounds, budget)}$

*Submission deadline: October 21th.

GECCO'13, July 6-10, 2013, Amsterdam, The Netherlands.
ACM ISBN TBA.
DOI: 10.1145/1235

- @param objective function provided by coco platform for test purpose.
- @param lower bounds of the search area representing the interesting region for the objective function.
- @param Upper bounds of the search area representing the interesting region for the objective function.
- @param The maximal number of evaluations(unless reaching the stopping criteria first).

The main loop consists of three main parts:
1. initialization of all the parameters properly
2. sampling of a fixed size of parents from the offsprings
3. update of the internal parameters($\sigma \in \mathbb{R}_+^n$,$s\_sigma \in \mathbb{R}$)

For concrete implementation details , we use matrix to calculate all the values for the high dimensions. $|\mathbb{N}(0,1)|$ is a half normal distribution, and its expectation equal to $\sqrt{2/\pi}$. And the value of $\mathbb{E}\,\|\mathbb{N}(0,\mathbb{I})\|$ depends on the dimension of the multi-variate normal distribution. But we can also approche the value by $\sqrt{n}(1-1/(4n)+1/(21n^2))$

$\boxed{sel\_u\_best(\mu, x_k, z_k)}$
- @param parental population size.
- @param new offsprings after mutation.
- @param mutation factor generated by an spherical (isotropic) distribution.

This function represents the recombination and parent update step, which decides the new parental population. The μ best offsprings are selected based on their fitness, here fitness is defined as the value of objective function. The smaller of the value generated by objective function, the better we consider the solution vector is. As a result, we ranked the index of all the offsprings within the size of μ. In the end, the function returns μ best solution vector as new parents and corresponding μ best mutation factor that will be used in the next step.

$\boxed{happy(x, x\_final)}$
- @param newly generated solution vector of a single parent
- @param last solution vector generated by last generation

Setting up a threshold which decides the stop criterion. When the difference of the last two result solutions is no bigger than the threshold, we consider the result reaches the convergence(happy condition).

unit test:

A simple objective function $f(x,y) = 5x^2 + 8y^2 + 15$ is used to do the algorithm unit test. Given limited budgets and search region, we are able to satisfy the happy condition within an acceptable time period.

Implementation on coco platform:

The default solver of the experience is modified as our algorithm which is referred as "*ES_search_path*". We add our own optimizer in the function def coco_optimize(solver, fun, max_evals, max_runs=e^9) and create the new branch as
elif solver.__name__ ==’ES_search_path’:
solver(fun, fun.lower_bounds,
fun.upper_bounds, remaining_evals)
Four parameters are respectively referred as objective function,lower bound of the search region, upper bound of the search region, remaining evaluation times. Through this interface, we executed our algorithm successfully on the coco platform.
All the source codes and benchmarking results are available on the site *github* under the project nbacxx23/ES-with-Search-Path. Please feel welcome to make a reference to the implementation and propose the useful advices for further improvements.

3.2 Parameters

- P a multiset of individuals, a population
- $\lambda \in \mathbb{N}$ number of off-springs
- $\mu \in \mathbb{N}$ number of parents, which approximates 1/4 λ
- $(lbounds, ubounds)$,search range $lbounds, ubounds \in \mathbb{R}$
- $\sigma \in \mathbb{R}_+^n$coordinate wise standard deviation (step size)
- $s\_sigma \in \mathbb{R}$ search path - carries info about the interrelation between single steps
- $x\_final \in \mathbb{R}^n$ the final solution vector after the optimization process
- $x\_k \in \mathbb{R}^n$ the feasible solutions of new parental candidates after mutation
- $E\_half\_normal\_dis$ expectation of half normal distribution
- E_muldim_normal estimation of the expectation of $\|N(0,I)\|$
- *budget* parameter given, the max number of loops

3.3 Stopping criteria

In our algorithm, there are two stopping criteria:
- The function **def happy(x,x_final)** tells if the precision criterion is satisfied (return 1) or not (return 0). If the distance between two consecutive searches is less than ϵ(in our case, this parameter is set as $10^(-9)$ but further experiences are possible and might be preferred), the function returns "happy". It makes judegement whether the solutions are convergence or not.
- The loop while stops if the **budget = 0**, which is the time stopping criterion to automatically avoid infinite iterations.

4. CPU TIMING

We have used the timing test code in example experiment code. For each dimension, the CPU time has been measured when running the algorithm on the benchmark suite. The time divided by the number of function evaluations. Here are our timing records:

Timing Experiments

Dimension	budget=2000	budget=5000	budget=10000
2	3.7e-05	1.6e-05	1.7e-05
3	3.4e-05	1.5e-05	2.6e-05
5	4.4e-05	1.5e-05	7.3e-05
10	5.1e-05	1.7e-05	2.0e-05
20	7.9e-05	2.3e-05	2.5e-05
40	1.4e-04	4.1e-05	3.8e-05

unit: $(seconds/evaluation)$

The test of budget 2000 is finished on a computer with AMD A6-3400 CPU, 1.4GHz, 4 cores; The budget 5000 is executed on a computer with intel Core i5 CPU, 2.7GHz, 2 cores; The budget 10000 is tested on a intel core i5 CPU, 2.4GHz
We can see that generally the higher the dimension is, the more time it takes for an evaluation.

5. RESULTS

Results from benchmarking ES-with-Search-Path and comparing it to BIPOP-CMA-ES and BFGS on the benchmark functions are presented in Figure [1-5] and Table[1].

The experiments were performed with COCO [2], version 1.0.1, the plots were produced with version 1.0.4

5.1 Analysis of average runtime

The average runtime(aRT) estimates the expected runtime of the restart algorithm. The formula is:

$$\text{aRT} = \frac{1}{n_\text{s}} \sum_i \text{RT}_i^\text{s} + \frac{1-p_\text{s}}{p_\text{s}} \frac{1}{n_\text{us}} \sum_j \text{RT}_j^\text{us} \quad (1)$$

$$= \frac{\sum_i \text{RT}_i^\text{s} + \sum_j \text{RT}_j^\text{us}}{n_\text{s}} \quad (2)$$

$$= \frac{\#\text{FEs}}{n_\text{s}} \quad (3)$$

where $p_\text{s} > 0$ is the probability of success of the algorithm with $n_\text{s} \geq 1$ successful runs with runtimes RT_i^s, and n_us unsuccessful runs with RT_j^us evaluations. Compared to the best average runtime, referring to Figure 2 and Table 1, from BBOB-2009 for $\Delta f = 10^{-8}$, all of our tests on 24 functions have a greater runtime than the best. In fact, for the f-target $f + \Delta f = 10^{-8}$, Our algorithm ran very well for all the separable functions $(f1 - f5)$, all of the trials succeeded to reach the target value. All the functions in all the dimensions with high conditioning and unimodal $(f10 - f14)$ ran badly,we couldn't reach the target value. The results from $f15 - f24$ weren't good except a few functions f15,f16,f21,f22, who have reached the target value.
Our algorithm ran well in low dimension and in the low scaling of Δf, especially for the separable functions $(f1 - f5)$ and low or moderate conditioning functions $(f6 - f9)$, it could reach the target value faster than the best aRT in 2009. When the dimension augmented, some functions couldn't reach target value.

5.2 Analysis of Empirical Cumulative Distribution Function of the runtime

To better measure the ES-with-Search-Path performance, the budget is fixed at 20000 for looking into the various hit target. Here COCO-platform provides with a very useful and interpretive plotting, which is called ECDF, referred to Figure 1. Empirical cumulative distribution functions (ECDF), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Left subplots: ECDF of the number of function evaluations (FEvals) divided by search space dimension D, to fall below $f_\text{opt} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. The thick red line represents the most difficult target value $f_\text{opt} + 10^{-8}$. Legends indicate for each target the number of functions that were solved in at least one trial within the displayed budget. Right subplots: ECDF of the best achieved Δf for running times of $0.5D, 1.2D, 3D, 10D, 100D, 1000D, \ldots$ function evaluations (from right to left cycling cyan-magenta-black...) and final Δf-value (red), where Δf and Df denote the difference to the optimal function value. Light brown lines in the background show ECDFs for the most difficult target of all algorithms benchmarked during BBOB-2009.

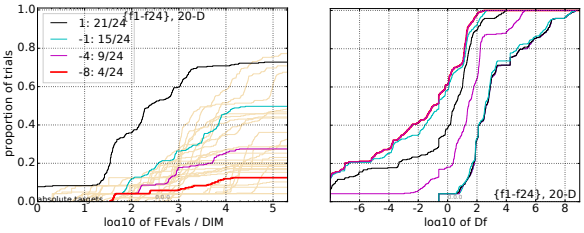


Figure 1: ECDF of all functions

However, it is difficult to measure a single algorithm. As is the case, the data sets of two baseline algorithms and another group are invited to compare with our results. All the data sets are plotted in Figure 3. It is the 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, 20, 40-D. The ?best 2009? line corresponds to the best aRT observed during BBOB 2009 for each selected target; the ? BIPOP-CMA? and ?BFGS? are two baseline algorithms ; and the ?ENSTA1? corresponds to the same algorithm by another group implemented with C++.

Comparison with baseline algorithms

As is shown in Figure 3, the blue line corresponds to ES-with-Search-Path results for all subgroups and all functions; the yellow line and the rose line represent the results of BIPOP-CMA-ES and BFGS; the light brown line corresponds to the best aRT observed during BBOB 2009 for each selected target.

Specially, the big cross on each line represents the end of runtime and shows the proportion of runs who hit the target. Given Budget=20000, our experiment stops at about 4.1, which is $O(10^4)$. According to the comparison of all functions, ES-with-Search-Path runs almost as well as BFGS but no better than BIPOP-CMA-ES. With ES-with-Search-Path , almost 67%,47%, 43% of runs can hit the tadget.

We interpret that, as to subgroup "separable functions", "multi-modal functions" and "weak structure functions", ES-with-Search-Path runs better than BFGS. As for the other groups, we are no better than the baseline group. That is to say, ES-with-Search-Path performs well for a certain type of problems.

Comparison with other group

As is shown in Figure 3, the light blue line represents the result of group "ENSTA1", who also implements ES-with-Search-Path but with C++. Thanks to the advantage of C++, "ENSTA1" has set the budget at 500000, which runs much more function evaluations. That is why their results are better predicted by COCO. Significantly, the results of us two groups is very close, which is a positive cross validation of the implementation. However, as for D=40, our result is much better then theirs, which may result from a overflow problem.

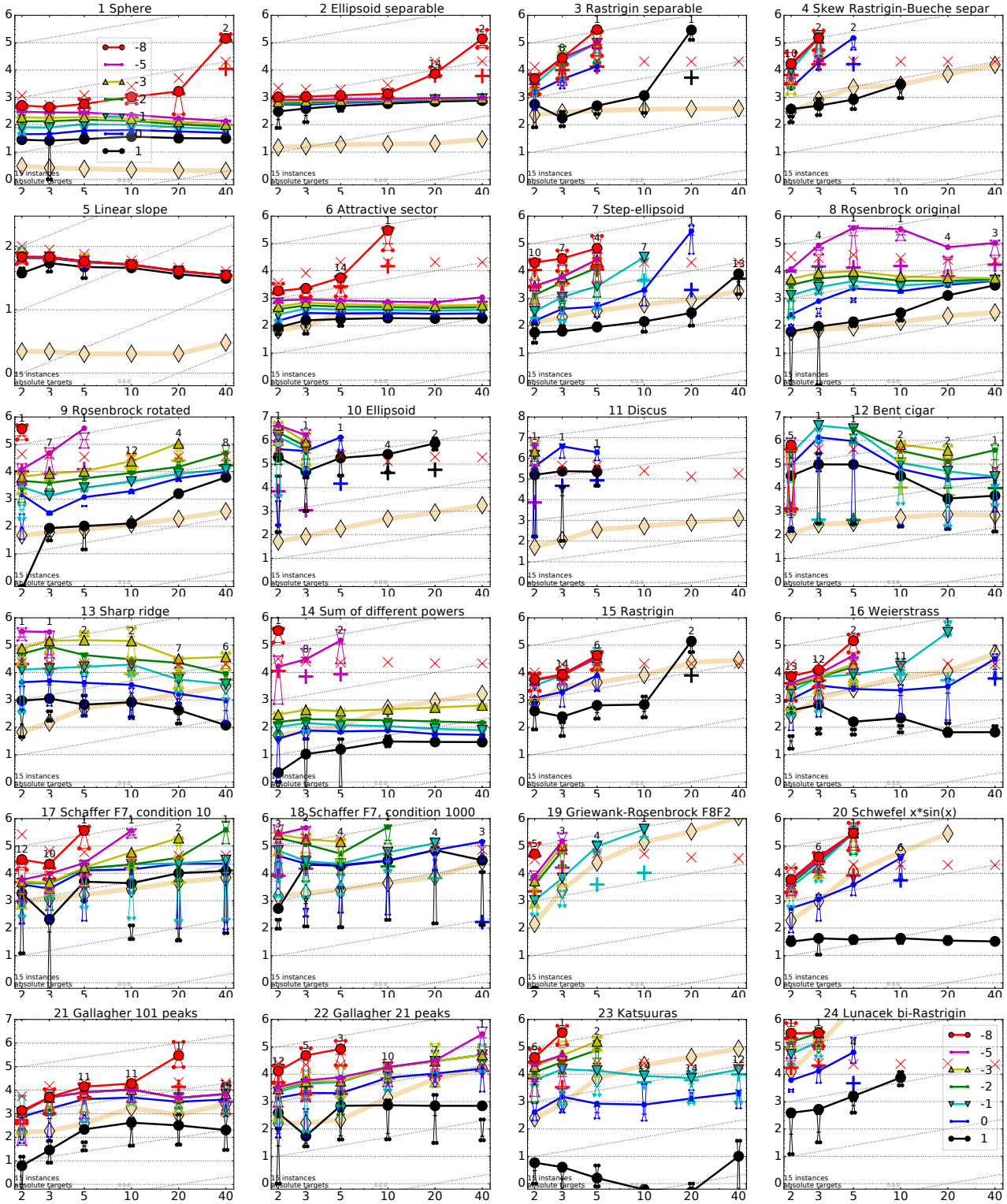


Figure 2: Scaling of runtime to reach $f_{\mathrm{opt}} + 10^{\#}$ with dimension; runtime is measured in number of f-evaluations and $\#$ is given in the legend; Lines: average runtime (aRT); Cross (+): median runtime of successful runs to reach the most difficult target that was reached at least once (but not always); Cross ($\times$): 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. Numbers above aRT-symbols (if appearing) indicate the number of trials reaching the respective target. The light thick line with diamonds indicates the respective best result from BBOB-2009 for $\Delta f = 10^{-8}$. Horizontal lines mean linear scaling, slanted grid lines depict quadratic scaling.

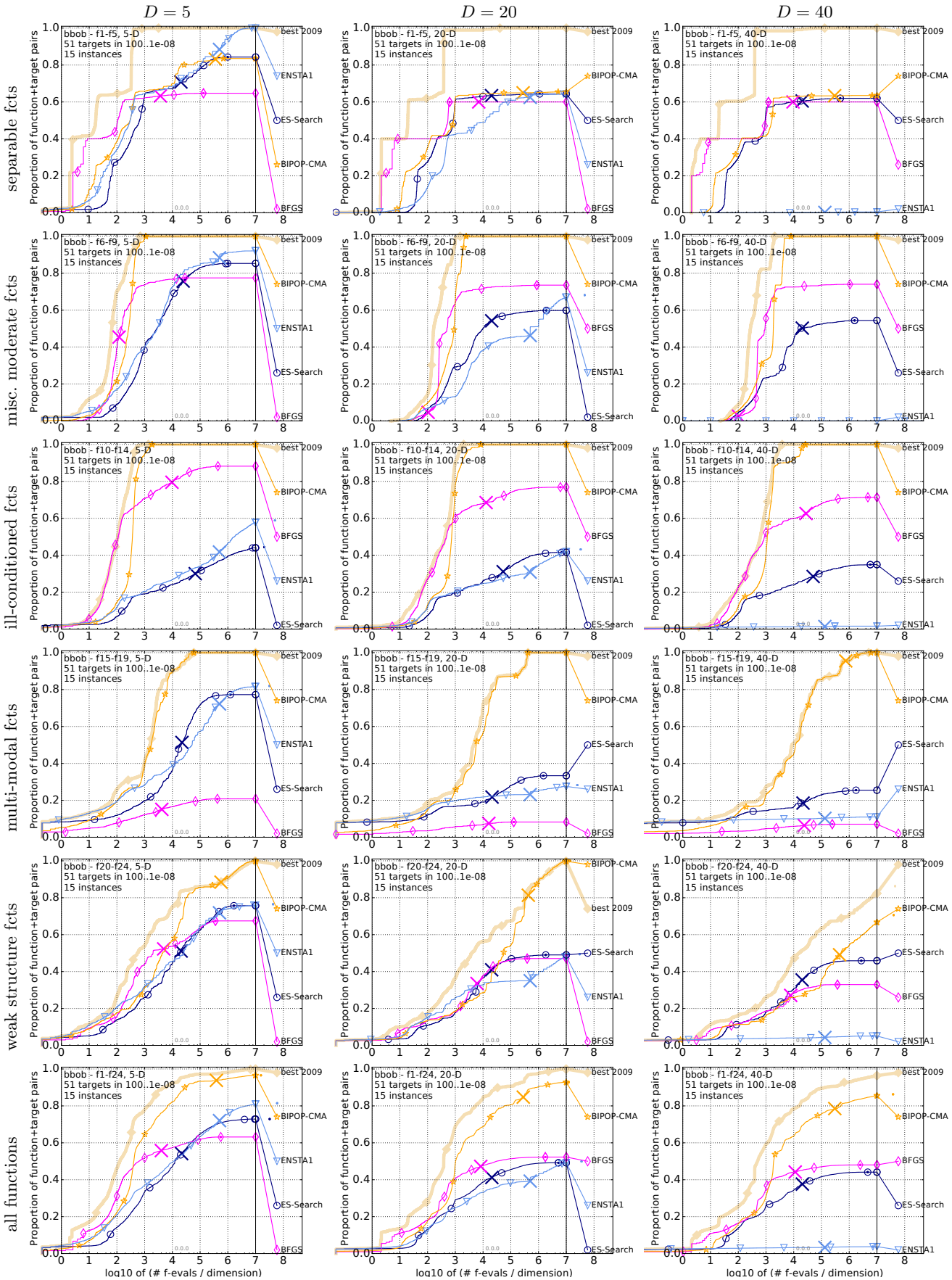


Figure 3: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5, 20, 40-D. The "best 2009" line corresponds to the best aRT observed during BBOB 2009 for each selected target; the " BIPOP-CMA" and "BFGS" are two baseline algorithms; and the "ENSTA1" corresponds to the same algorithm by another group implemented with C++.

5-D

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
	14(9)	25(9)	43(22)	62(4)	79(21)	114(15)	164(17)	15/15
f_2	83	87	88	89	90	92	94	15/15
	29(14)	36(5)	38(4)	41(3)	43(3)	47(4)	53(16)	15/15
f_3	716	1622	1637	1642	1646	1650	1654	15/15
	3.4(2)	42(62)	290(131)	290(265)	289(171)	289(549)	288(362)	1/15
f_4	809	1633	1688	1758	1817	1886	1903	15/15
	5.1(3)	456(593)	∞	∞	∞	∞	∞ *1.0e5*	0/15
f_5	10	10	10	10	10	10	10	15/15
	24(7)	28(6)	29(7)	29(6)	29(5)	29(6)	29(6)	15/15
f_6	114	214	281	404	580	1038	1332	15/15
	7.6(4)	6.5(2)	6.8(1)	6.1(1)	5.2(0.9)	4.0(0.6)	5.4(4)	14/15
f_7	24	324	1171	1451	1572	1572	1597	15/15
	19(9)	7.6(9)	12(2)	44(41)	82(113)	82(62)	108(61)	4/15
f_8	73	273	336	372	391	410	422	15/15
	9.3(5)	42(2)	63(188)	90(93)	122(3)	4493(3317)	∞ *1.2e5*	0/15
f_9	35	127	214	263	300	335	369	15/15
	15(9)	46(179)	60(18)	108(41)	172(139)	5720(3133)	∞ *1.3e5*	0/15
f_{10}	349	500	574	607	626	829	880	15/15
	2605(4165)	13616(10472)	∞	∞	∞	∞	∞ *3.6e5*	0/15
f_{11}	143	202	763	977	1177	1467	1673	15/15
	7864(7926)	48890(84772)	∞	∞	∞	∞	∞ *4.7e5*	0/15
f_{12}	108	268	371	413	461	1303	1494	15/15
	4442(8204)	17556(18815)	42363(25714)	38015(26413)	∞	∞	∞ *9.9e5*	0/15
f_{13}	132	195	250	319	1310	1752	2255	15/15
	25(26)	108(147)	324(571)	678(1036)	581(746)	∞	∞ *1.0e5*	0/15
f_{14}	10	41	58	90	139	251	476	15/15
	8.0(13)	8.7(6)	10(5)	10(2)	14(3)	2949(7548)	∞ *1.1e5*	0/15
f_{15}	511	9310	19369	19743	20073	20769	21359	14/15
	6.2(6)	4.2(5)	8.8(11)	8.6(10)	8.5(12)	8.3(6)	8.1(9)	6/15
f_{16}	120	612	2662	10163	10449	11644	12095	15/15
	6.6(5)	20(28)	16(10)	7.8(8)	10(7)	18(20)	38(81)	2/15
f_{17}	5.2	215	899	2861	3669	6351	7934	15/15
	4773(1759)	297(418)	83(146)	27(60)	22(24)	19(18)	32(37)	1/15
f_{18}	103	378	3968	8451	9280	10905	12469	15/15
	911(549)	267(20)	28(10)	31(58)	76(83)	∞	∞ *1.1e5*	0/15
f_{19}	1	1	242	1.0e5	1.2e5	1.2e5	1.2e5	15/15
	1	1	1991(1564)	∞	∞	∞	∞ *1.6e5*	0/15
f_{20}	16	851	38111	51362	54470	54861	55313	14/15
	12(4)	23(26)	39(49)	29(29)	27(21)	27(37)	27(21)	1/15
f_{21}	41	1157	1674	1692	1705	1729	1757	14/15
	26(34)	18(19)	24(47)	24(34)	24(19)	24(24)	23(22)	11/15
f_{22}	71	386	938	980	1008	1040	1068	14/15
	49(106)	26(44)	28(45)	27(37)	28(47)	37(15)	136(99)	3/15
f_{23}	3.0	518	14249	27890	31654	33030	34256	15/15
	2.7(2)	8.1(11)	4.7(4)	14(10)	25(12)	∞	∞ *1.1e5*	0/15
f_{24}	1622	2.2e5	6.4e6	9.6e6	9.6e6	1.3e7	1.3e7	3/15
	4.9(6)	1.5(1)	∞	∞	∞	∞	∞ *1.1e5*	0/15

20-D

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
	15(4)	26(6)	37(3)	48(8)	59(6)	80(4)	115(5)	15/15
f_2	385	386	387	388	390	391	393	15/15
	36(2)	41(4)	43(6)	44(3)	45(3)	47(3)	66(3)	14/15
f_3	5066	7626	7635	7637	7643	7646	7651	15/15
	1136(1036)	∞	∞	∞	∞	∞	∞ *4.0e5*	0/15
f_4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
	∞	∞	∞	∞	∞	∞	∞ *4.0e5*	0/15
f_5	41	41	41	41	41	41	41	15/15
	18(1)	20(1)	20(0.9)	20(2)	20(2)	20(2)	20(2)	15/15
f_6	1296	2343	3413	4255	5220	6728	8409	15/15
	2.8(0.5)	2.3(0.4)	2.1(0.3)	2.1(0.3)	2.0(0.2)	2.1(0.3)	67(35)	0/15
f_7	1351	4274	9503	16523	16524	16524	16969	15/15
	4.3(2)	1324(1104)	∞	∞	∞	∞	∞ *4.0e5*	0/15
f_8	2039	3871	4040	4148	4219	4371	4484	15/15
	12(4)	16(7)	18(2)	21(12)	27(3)	333(232)	∞ *4.6e5*	0/15
f_9	1716	3102	3277	3379	3455	3594	3727	15/15
	18(2)	37(33)	51(3)	89(41)	620(449)	∞	∞ *5.8e5*	0/15
f_{10}	7413	8661	10735	13641	14920	17073	17476	15/15
	2048(2046)	∞	∞	∞	∞	∞	∞ *1.8e6*	0/15
f_{11}	1002	2228	6278	8586	9762	12285	14831	15/15
	∞	∞	∞	∞	∞	∞	∞ *1.3e6*	0/15
f_{12}	1042	1938	2740	3156	4140	12407	13827	15/15
	66(0.1)	229(255)	364(512)	859(1474)	1823(1345)	∞	∞ *7.0e5*	0/15
f_{13}	652	2021	2751	3507	18749	24455	30201	15/15
	13(9)	16(24)	40(38)	128(117)	34(33)	∞	∞ *4.1e5*	0/15
f_{14}	75	239	304	451	932	1648	15661	15/15
	7.8(2)	4.8(2)	5.7(1)	7.2(1)	11(1)	∞	∞ *4.2e5*	0/15
f_{15}	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
	92(110)	∞	∞	∞	∞	∞	∞ *4.1e5*	0/15
f_{16}	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
	0.94(0.8)	2.3(4)	76(75)	∞	∞	∞	∞ *4.1e5*	0/15
f_{17}	63	1030	4005	12242	30677	56288	80472	15/15
	3216(3696)	434(516)	116(161)	61(43)	127(96)	∞	∞ *6.0e5*	0/15
f_{18}	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
	2278(4119)	357(304)	129(98)	∞	∞	∞	∞ *6.6e5*	0/15
f_{19}	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
	1	1	∞	∞	∞	∞	∞ *4.9e5*	0/15
f_{20}	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
	8.5(2)	∞	∞	∞	∞	∞	∞ *4.0e5*	0/15
f_{21}	561	6541	14103	14318	14643	15567	17589	15/15
	11(17)	10(15)	6.6(10)	6.5(13)	6.4(6)	6.1(4)	6.6(11)	1/15
f_{22}	467	5580	23491	24163	24948	26847	1.3e5	12/15
	30(43)	38(63)	25(42)	25(23)	24(33)	23(24)	∞ *4.0e5*	0/15
f_{23}	3.2	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
	3.1(5)	16(15)	2.1(1)	∞	∞	∞	∞ *4.2e5*	0/15
f_{24}	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
	∞	∞	∞	∞	∞	∞	∞ *4.3e5*	0/15

Table 1: Average running time (aRT in number of function evaluations) divided by the best aRT measured during BBOB-2009. 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 in the first. The different target Δf-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\text{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Bold entries are statistically significantly better (according to the rank-sum test) compared to the best algorithm in 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.

5.3 Analysis of aRT loss ratio

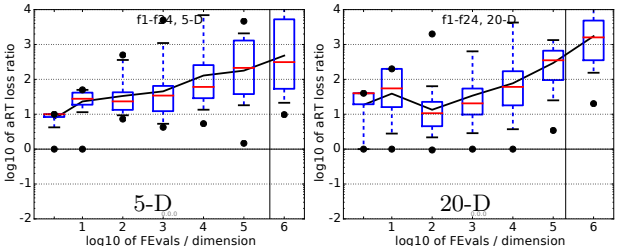
From aRT loss ratio graph of all functions in 5D, we found that the aRT loss ratio augmented evidently following the increase of numbers of f-evaluations. The increase of f-evaluations represented the rate of unsuccessful reaching to the target value. In 20-D, the successful rate augmented when the number of the f-evaluations was low. Our algorithm ran well for the low scaling precision and got a higher success rate for reaching target values. So when the number of f-evaluations is low, we have a lower loss ratio.

6. CONCLUSION

During the experiment, we encountered several problems. First of all, we could not find the optimal solution of problem. After rechecking our code, we noticed that we included all of the mutation factors in the equation 7b of the offspring. The recombination and selection procedure is crucial in guaranteeing the promising optimization direction. However, without the selection procedure, the solution vector tends to more possible to get out of the search range since the step size of the next generation is decided by the average value of the mutation factor, which in our case is increased by false formula. As a result, the worse factors affect the covariance of the normal distribution in the next step. When the experience dimension reaches 40, the values of vector will become too large to keep in the normal region and keeps showing up the overflow problem on Coco. After we rewrite the algorithm and run the test again, it succeeds on both the unit test and coco platform.

After that, we also encountered the problem during the initialization step. With the help of professor, we know that our purpose is to make sure that we can get the points in the search region with a large probability. So 40 percent to 60 percent of the search region should be covered by the 3σ domain of the normal distribution. It ensures that our values are reasonable and also avoided the overflow problem.

Through this project, we get familiar with the black-box optimization problem and succeed in implementing an algorithm on a benchmarking platform. Since the evolution strategy is inspired from the biological area, it is important to pay attention to the development of other research areas. Usually, other subjects can provide some creative ideas in computer science or more specifically, optimization problem.



#FEs/D	best	10%	25%	**med**	75%	90%
	f_1–f_{24} **in 5-D**, maxFE/D=440000					
2	1.0	3.1	8.3	10	10	10
10	1.0	9.9	18	28	42	50
100	7.3	9.2	13	23	44	4.3e2
1e3	4.2	4.9	11	34	67	1.6e3
1e4	5.4	12	29	61	2.7e2	1.0e4
1e5	1.5	17	37	2.1e2	1.4e3	2.3e3
1e6	9.7	19	48	3.1e2	7.1e3	1.5e4
$\mathrm{RL_{US}}$/D	2e4	2e4	2e4	2e4	3e4	9e4
	f_1–f_{24} **in 20-D**, maxFE/D=203660					
#FEs/D	best	10%	25%	**med**	75%	90%
2	1.0	1.0	17	40	40	40
10	1.0	1.0	13	55	2.0e2	2.0e2
100	0.94	2.0	4.3	11	24	2.6e2
1e3	1.0	2.7	9.2	21	58	2.6e3
1e4	1.0	2.9	16	61	1.8e2	7.1e3
1e5	3.4	19	90	3.5e2	6.8e2	1.8e3
1e6	20	1.1e2	3.5e2	1.6e3	5.0e3	1.8e4
$\mathrm{RL_{US}}$/D	2e4	2e4	2e4	2e4	3e4	5e4

Figure 4: aRT **loss ratio versus the budget in number of f-evaluations divided by dimension. For each given budget** FEvals**, the target value f_t is computed as the best target f-value reached within the budget by the given algorithm. Shown is then the** aRT **to reach f_t for the given algorithm or the budget, if the GECCO-BBOB-2009 best algorithm reached a better target within the budget, divided by the best** aRT **seen in GECCO-BBOB-2009 to reach f_t. Line: geometric mean. Box-Whisker error bar: 25-75%-ile with median (box), 10-90%-ile (caps), and minimum and maximum** aRT **loss ratio (points). The vertical line gives the maximal number of function evaluations in a single trial in this function subset. See also Figure 5 for results on each function subgroup.**

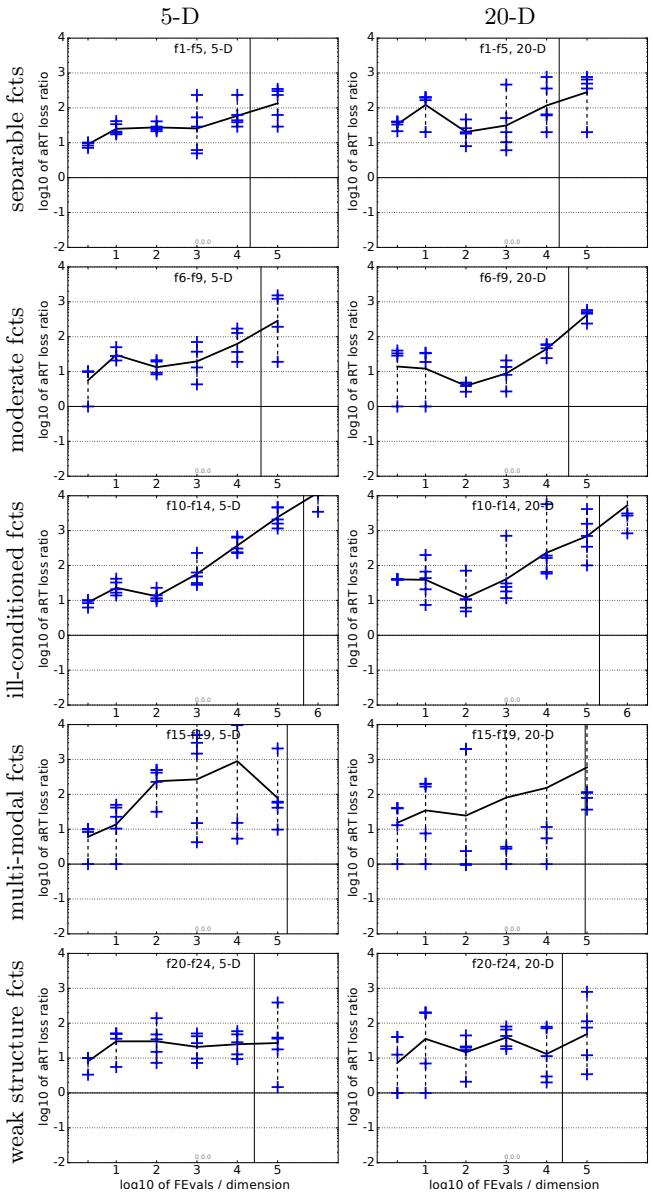


Figure 5: aRT **loss ratios (see Figure 4 for details). Each cross (+) represents a single function, the line is the geometric mean.**

7. REFERENCES

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

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

[3] N. Hansen, T. Tušar, O. Mersmann, A. Auger, and D. Brockhoff. COCO: The experimental procedure. *ArXiv e-prints*, arXiv:1603.08776, 2016.