The 2019 100-Digit Challenge on Real-Parameter, Single Objective Optimization: Analysis of Results

K. V. Price[1], N. H. Awad[2], M. Z. Ali[3], P. N. Suganthan[4]

[1] Vacaville, California, USA

[2]Department of Computer Science, University of Freiburg, Freiburg, Germany

[3]School of Computer Information Systems, Jordan University of Science and Technology, Jordan

[4]School of EEE, Nanyang Technological University, Singapore

kvprice@pacbell.net, awad@cs.uni-freiburg.de, mzali.pn@ntu.edu.sg, epnsugan@ntu.edu.sg

July 2019

Updated December 21, 2019

***Abstract*—This report presents and discusses the results of the 100-Digit Challenge on real-parameter single objective numerical optimization. It includes results from the 2019 Congress on Evolutionary Computation (CEC2019), the 2019 Genetic and Evolutionary Computation Conference (GECCO2019) and the 2019 Swarm, Evolutionary and Memetic Computing Conference (SEMCCO2019). The top four codes have been verified.**

TABLE 1.
ALGORITHMIC ACRONYMS IN THIS REPORT

Acronym	Algorithm
ABC	Artificial Bee Colony
CIPDE	Collective Information Powered Differential Evolution
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CMEAL	Cooperative Model of Evolutionary Algorithms
Co-Op	Cooperative Optimization
DE	Differential Evolution
DISH	DIstance-based parameter adaptation for Success-History based DE
DISHchain1e+12	DIstance-based Success History DE for large populations and long run times
DLABC	Division of Labor Artificial Bee Colony
DSFABC_elite	Depth Search First ABC with elite guiding
EBOwithCMAR	Effective Butterfly Optimizer with Covariance Matrix Adapted Retreat phase
ELSHADE-SPACMA	Enhanced LSHADE with Semi-Parameter Adaptation hybrid with CMA-ES
ESHADE	Enhanced SHADE
ESHADE-USM	Enhanced SHADE with Univariate Sampling Method
ESP-SOMA	Ensemble of Strategies and Perturbation parameter in Self-Organizing Migrating Algorithm
EA	Evolutionary Algorithm
GA	Genetic Algorithm
GADE	Genetic Algorithm with DE
HS	Harmony Search
HS-ES	Hybrid Sampling Evolution Strategy
HTPC	Hybrid Time window Particle swarm optimization and Covariance matrix adaptation
HyDE-DF	Hybrid DE with Decay Function
iL-SHADE	Improved L-SHADE
jDE100	J? DE for the 100 digit challenge
jSO	A modified LSHADE algorithm
lbestPSO	Local best Particle Swarm Optimization
LS	Local Search
LSHADE	Success History-based Adaptation DE with Linear population size reduction
LSHADE_cnEpSin	An ensemble sinusoidal parameter adaptation incorporated with L-SHADE
LSHADE_RSP	LSHADE algorithm with Rank-based Selective Pressure strategy
MiLSHADE-LSP	Memetic algorithm with improved L-SHADE and a Local Search Pool
mL-SHADE	Modified L-SHADE
PSO	Particle Swarm Optimization
rCIPDE	Restart-based Collective Information Powered DE
rjDE	Restart jDE
SHADE	Successful History-based Adaptive DE
SOMA Pareto	Pareto-based Self-Organizing Migrating Algorithm
SOMA T3A	Team-To-Team Adaptive Self-Organizing Migrating Algorithm
TW-PSO	Time Window PSO
UNIVAR	UNIVARiate sampling method
USM	Univariate Sampling Method

I. Introduction

Æsop's fabled race between Tortoise and Hare is a cautionary tale about relying too much on speed as a criterion for success. Recall that after quickly outpacing Tortoise and feeling confident of his impending victory, Hare pauses by the side of the road, but then falls asleep. When he awakes, Hare sees Tortoise nearing the finish-line and even though he redoubles his efforts, Hare still loses to the slower, but more persistent Tortoise. The moral of the story is: "The race is not always to the swift".

A similar situation exists when testing optimization algorithms. Like Hare who fell asleep, aggressive search algorithms may quickly find themselves languishing in a non-global basin of attraction. On the other hand, simulated annealing—like Tortoise who is slow but persistent—*in theory* can optimize any function when given enough time [1]. The burning question, of course, is how much time?

Figure 1 shows convergence plots (function value *vs*. function evaluations) for two algorithms. The objective function and the algorithms are hypothetical, but the relation between the two plots should look familiar. If we terminate trials after FE_1 function evaluations, then algorithm 1 (Hare) will be deemed the better algorithm, i.e. the one producing the lower objective function value. If, however, we terminate trials after FE_2 function evaluations, then algorithm 2 (Tortoise) wins. Clearly, the better algorithm in this case depends on when trials are terminated. It is with the goal of minimizing the dependence of measured performance on function evaluation limits and instead focusing on solution accuracy that we have developed the 100-Digit Challenge on real-parameter, single objective optimization.

After a quick look at the story behind the original 100-Digit Challenge, we offer a rationale for the competition, followed by section II, which sets forth and discusses the rules of the competition and some issues that arose around them. Section III presents the competition's results, which are discussed in section IV. Section V concludes this report with a look at what we learned. Plots of the competition's results appear in an Appendix.

A. Background

The original 100-Digit Challenge was created in 2002 by Oxford's Nick Trefethen in conjunction with the Society for Industrial and Applied Mathematics (SIAM) as a test for high-accuracy computing [2], [3]. Specifically, the challenge was to solve 10 hard problems to 10 digits of accuracy. One point was awarded for each correct digit, making the maximum score 100; hence the name. Contestants were allowed to apply any method to any problem and take as long as necessary to solve it. Out of the 94 teams that entered, 20 scored 100 points and 5 others scored 99.

The 100-Digit Challenge for single objective, real-parameter optimization resembles the original SIAM version in that we ask contestants to solve (optimize) ten functions to an accuracy of 10 digits each. As in the original challenge, one point is awarded for each correct digit, so the maximum total score for all ten functions is again 100. We also allowed contestants as much time as necessary to get a good result. Different from the SIAM version, our rules require that contestants optimize all ten functions with a single algorithm, not any method for any function. To restore some of the original contest's flexibility, however, we have allowed contestants to tune up to two algorithmic parameters.

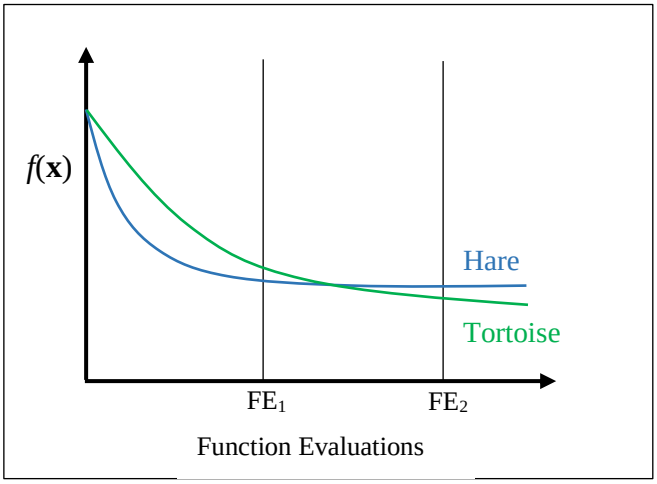


Fig. 1. The better algorithm depends on when they are compared.

B. Rationale

Most real parameter optimization competitions measure the objective function value at multiple "vertical" time-slices of the convergence plot, like in Fig. 1. The 100-Digit Challenge complements this approach by measuring function values at multiple "horizontal" slices of the convergence plot. In some earlier contests, the "horizontal slice" was a given "value-to-reach" (VTR) and the average number of function evaluations to attain the VTR was the primary statistic of performance. Previous contests have also relied on the success performance measure. One problem with this approach is that it depends on the notion of a success, which is undefined when an algorithm fails to attain the VTR. The 100-Digit Challenge, however, provides a more graduated way to measure "horizontal" performance (accuracy) because even "failures" can have some correct digits.

Why take this alternative approach? The goal of global optimization is to locate a global optimum, not just to see how close an algorithm can get to the optimum in an arbitrary amount of time, yet competitions often do not allow enough time for an algorithm to find a global optimum. When exploring algorithmic performance, it is better to see if an algorithm can locate the optimum (more technically, the basin containing the optimum) and then find out how long it took. In this view, the number of objective function evaluations is a measurement, not a constraint. This approach provides all of the information in time-limited studies and shows the resources needed to globally optimize these functions.

There is also a sense in which limits on accuracy are less arbitrary than time limits. With test functions, we usually know how many digits of accuracy are necessary to ensure that a solution lies near the optimum in the global basin of attraction, but we don't know how long it will take. Similarly, we really don't need, e.g. 20 significant digits of accuracy for real-world problems, because solutions that accurate cannot be implemented to such a high degree of precision. Furthermore, finding additional digits once an algorithm is close to the optimum in a global basin of attraction usually takes comparatively little time, i.e. there is little penalty for over specifying the level of accuracy. Given this, ten digits seems a reasonable choice—sufficient, but not excessive.

II. Contest Details

A. Functions

Table II lists the basic functions in this year's competition. All functions are designed to have a minimum value of 1.0. Functions F4, F5, F6, F7, F8 and F10 were familiar to most entrants, having been staples of prior CEC and GECCO competitions. To ensure complete parameter dependence, these basic functions are rotated (and shifted). A full description of all ten test functions can be found at [4]: https://github.com/P-N-Suganthan.

Functions F1, F2, F3 and F9 were new to many contestants. Functions F1, F2, F3 are fully parameter dependent and not rotated (or shifted). Storn's Chebyshev function (F1) is a classical optimization problem whose solution was once considered to be exceptionally hard to find. In this implementation, F1 is an error function that regularly samples the polynomial implied by the vector being evaluated. The solution is highly conditioned, integral and every other parameter value is zero. For example, the solution for $D = 9$ is:

$$128, 0, -256, 0, 160, 0, -32, 0, 1$$

The extreme conditioning of this function becomes more apparent as the dimension increases. For example, here is the solution for $D = 17$:

$$32768, 0, -131072, 0, 212992, 0, -180224, 0, 84480, 0, -21504, 0, 2688, 0, -128, 0, 1.$$

With Chebyshev, you can get the required 10-digit accuracy without the parameter values being exact. This is less the case for the Inverse Hilbert Matrix Problem (F2). Like Chebyshev, F2 is both highly conditioned and implemented as an error function, but F2 requires a high precision format (long double for $D = 36$) and very accurate parameter values to reach 10 digits of accuracy in function value. Table III shows the solution for $D = 16$. We expected Differential Evolution (DE)-based algorithms to do well on these two functions, but were interested to see how other methods would fair coping with such highly conditioned landscapes.

TABLE II.

THE 100-DIGIT CHALLENGE BASIC TEST FUNCTIONS

No.	Functions	$F_i^* = F_i(\mathbf{x}^*)$	D	Search Range
1	Storn's Chebyshev Polynomial Fitting Problem	1	9	[−8192, 8192]
2	Inverse Hilbert Matrix Problem	1	16	[−16384, 16384]
3	Lennard-Jones Minimum Energy Cluster	1	18	[−4,4]
4	Rastrigin's Function	1	10	[−100,100]
5	Griewangk's Function	1	10	[−100,100]
6	Weierstrass Function	1	10	[−100,100]
7	Modified Schwefel's Function	1	10	[−100,100]
8	Expanded Schaffer's F6 Function	1	10	[−100,100]
9	Happy Cat Function	1	10	[−100,100]
10	Ackley Function	1	10	[−100,100]

The Lennard-Jones function, F3, mimics the potential energy of atomic interactions, being negative at large scales and becoming positive when atoms are separated by less than a certain distance. The optimization problem is to find an arrangement of "Lennard-Jonesium" atoms whose potential energy is a minimum. The problem is simpler than its 18 dimensions indicate, since only 6 atoms are being arranged and arranging 4 is trivial. Additionally, optimal parameter values are not unique because the location and orientation of the cluster are not constrained, which makes the problem effectively 15-dimensional. Erroneous results can occur when computing the Lennard-Jones potential if two atoms are placed in the same location (as can happen with certain forms of crossover). The energy contribution from the overlapping atoms is infinite, which is not a number and therefore typically not added to the total energy (function value), making the computed energy of the cluster appear to be lower than the known minimum. We addressed this problem by setting F3's function value to 1.0e+20 for any pair of atoms closer than 1.0e−20.

This year's contest also introduced the Happy Cat [5] function (F9), which was designed to be difficult for direct searches. The name comes from the fact that some contours of the two-dimensional version of this function resemble a smiling cat. In two-dimensions, an algorithm must navigate a circular trench with steep walls that is only slightly tilted in the direction of the optimum. This function was also shifted and rotated.

TABLE III.
THE SOLUTION TO THE INVERSE HILBERT FUNCTION FOR $D = 16$

16	−120	240	−140
−120	1200	−2700	1680
240	−2700	6490	−4200
−140	1680	−4200	2800

B. Scoring

An algorithm's score for a given function is the average number of correct digits in the best 25 of 50 trials. The idea that we would only count the best 25 of 50 trials to compute an algorithm's score for a given function confused some people (mostly reviewers!). First, we chose 50 trials because such a large sample makes it difficult to cherry-pick results. If, however, the score for a function had included results from all 50 trials, then all 50 trials would need to find all ten digits to score 10 on a given function. Unless the function is very simple (like the "sphere"), getting an evolutionary algorithm to find the optimum to ten digits of accuracy in each of 50 *consecutive* trials usually requires a *very* large population. In practice, it is often faster to run several trials with a smaller population at a lower rate of success than it is to run one trial with a population large enough to virtually

ensure that every trial finds all ten digits. By settling on the 50% rate of success to earn a perfect score of 10, we had hoped to get population sizes that more accurately represent what an algorithm requires in practice.

Upon seeing the first entries, it was immediately clear that we should have been more specific about how results should be formatted. The results of the 50 runs are to be sorted according to the *final* number of correct digits that each trial found. A large percentage of entrants misinterpreted this to mean, for example, that if a trial found 10 correct digits, then it also found the other 9, and so each cell 1–10 would be incremented, not just the final cell (Table IV). The correct format (Table V) is to enter just the final number of correct digits for each trial, which makes the *sum for a row equal to 50*. Also, the results for all 50 (not just the best 25) trials should be entered.

TABLE IV
INCORRECT FORMATTING FOR THE NUMBER OF CORRECT DIGITS

Function	Number of correct digits											Score
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
1	50	50	50	50	50	50	50	50	50	50	50	10
2	50	50	50	50	50	50	50	50	50	46	20	9.8

TABLE V
CORRECT FORMATTING FOR THE NUMBER OF CORRECT DIGITS. ROW SUM = 50.

Function	Number of correct digits											Score
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
1	0	0	0	0	0	0	0	0	0	0	50	10
2	0	0	0	0	0	0	0	0	4	26	20	9.8

C. Tuning

Given that there was no time limit, i.e. no limit on the maximum allowed number of functions evaluations, it made sense that the time taken for tuning should not count, so we allowed it, but since the tuning effort exponentially increases with the number of algorithmic control parameters (and itself becomes an optimization problem), we limited tuning to just 2 parameters (the same two for all functions), presuming that in most cases, population size would be one of the tuned parameters. Adaptive parameters were not counted as tuned parameters.

III. RESULTS

With the goal of encouraging "cross-pollination" between three of the largest conferences on evolutionary computation, we offered the 100-Digit Challenge at CEC-2019, GECCO-2019 and SEMCCO-2019 and have combined the results in this report. In total, 13 papers were accepted into the CEC proceedings, 3 abstracts were accepted at GECCO and 3 papers were accepted for SEMCCO for a total of 18 unique *primary* (i.e. possibly tuned) algorithms (not 19 because there were two SOMA T3A submissions, one at GECCO and another at SEMCCO). Some submissions, however, included results for more than one algorithm. In all, 20 additional algorithms were tested, with most being among the better performing algorithms at prior, time-limited competitions. By and large, these *secondary* algorithms were not tuned, but run with previously published control parameter defaults. In the next section, we explore a taxonomy of algorithms based on their core strategies, while section *B* presents scores for primary algorithms and declares a winner. Section *C* presents scores for the secondary algorithms.

A. Taxonomy of Primary Algorithms

Table VI is a taxonomy of primary algorithms based on their core generating strategies. The number next to an algorithm's acronym is the reference number of the paper in which the contest result appears. Perhaps the most obvious feature in Table VI is the dominance of DE-based algorithms. Only the three self-organizing migrating algorithms (SOMA, SOMA T3A and SOMA Pareto), an artificial bee colony algorithm (DLABC) and a particle swarm optimization-covariance matrix adaptation hybrid (HTPC) eschewed DE.

Table VI also makes clear that many entrants took a memetic approach. In addition to the seven algorithms that enlisted more than one core strategy, several others relied on multiple versions of a single core strategy. In particular, UMDE-MS included three DE-based generating strategies to augment the univariate sampling method. Similarly, ESP-SOMA took a memetic approach by calling three different SOMA strategies. Likewise, CMEAL employed three different DE strategies/algorithms in conjunction with CMA. ESHADE-USM drew from four DE mutation/recombination strategies.

Most primary algorithms also adaptively adjusted control parameters. The most commonly self-adapted variables were the DE mutation scale factor *F* and binomial crossover parameter *Cr*. Here, "self-adaptation" means that each member of the population evolves its own values for these two parameters. It is not meant to imply that parameters are part of the genome (evolving vectors). Most of these algorithms adopted the jDE approach to self-adapting both *F* and *Cr*, usually within the context of a SHADE-based algorithm. Two other algorithms, DISH and DISHchain1e+12 also self-adapted both *F* and *Cr*, but unlike jDE which adapts parameters based on feedback from function values, adaptation in the DISH algorithms is based on a distance measure (the distance between DE's target and trial vectors).

DE parameters were not the only adaptive variables. There is, of course, the adaptive covariance matrix whose elements are typically derived from a distribution of better trial vectors, but more straightforward examples of parameter adaptation can be found in the ESP-SOMA, SOMA T3A and DLABC algorithms.

TABLE VI.
DISTRIBUTION OF CORE STRATEGIES

Primary Algorithm[REF]	Core Strategy								
	ABC	CMA	DE	GA	HS	LS	PSO	SOMA	USM
UMDE-MS[6]			●						●
DISH[7]			●						
MiLSHADE-LSP[8]			●			●			
rCIPDE[9]			●						
jDE100[10]			●						
HTPC[11]		●					●		
ESHADE-USM[12]			●			●			●
CMEAL[13]		●	●						
ESP-SOMA[14]								●	
Co-Op[15]			●		●		●		●
mL-SHADE[16]			●						
DLABC[17]	●								
GADE[18]			●	●					
HyDE-DF[19]			●						
SOMA T3A[20, 23]								●	
DISHchain1e+12[21]			●						
SOMA Pareto[22]								●	
rjDE[24]			●						

B. Primary Algorithm Scores…and the Winner is…

Table VII shows the scores for each of the primary algorithms for all ten functions. Only jDE100 was able to attain the perfect score of 100. DISHchain1e+12 came in a close second with 97.12 points. The only tie was for third place, with both HyDE-DF and SOMA-T3A scoring 93. The lowest score was 51.92 (ESP-SOMA). The average score among primary algorithms was 80.1133.

The top 10 primary algorithms all scored 10 on functions F1–F7 and F10 (except for Co-Op which scored 9.56 on F7). Consequently, only their performance on F8 and F9 distinguished these top-tier algorithms from one another. Finding all 10 digits

for F8 put four algorithms ahead of the rest. The performance of these four algorithms on the Happy Cat function (F9) is what distinguished one from one another, except for SOMA-T3A and HyDE-DF, which tied with 3 digits each. Only jDE100 found all 10 digits, but the average of 7.12 digits found by DISHchain1e+12 for F9 was significantly more than any other algorithm except jDE100, with the remaining algorithms finding 4 or fewer digits.

TABLE VII.
SCORES FOR 18 PRIMARY ALGORITHMS

Primary Algorithm[REF]	Score by Function										Total Score
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	
jDE100[10]	10	10	10	10	10	10	10	10	10	10	**100**
DISHchain1e+12[21]†	10	10	10	10	10	10	10	10	7.12	10	**97.12**
HyDE-DF[19]	10	10	10	10	10	10	10	10	3	10	**93**
SOMA T3A[23]	10	10	10	10	10	10	10	10	3	10	**93**
ESHADE-USM[12]	10	10	10	10	10	10	10	2	3.52	10	**85.52**
SOMA Pareto[22]‡	10	10	10	10	10	10	10	2	3.04	10	**85.04**
rCIPDE[9]	10	10	10	10	10	10	10	2	3	10	**85**
Co-Op[15]	10	10	10	10	10	10	9.56	1	4	10	**84.56**
DISH[7]	10	10	10	10	10	10	10	1.92	2	10	**83.92**
rjDE[24]	10	10	10	10	10	10	10	1.52	2	10	**83.52**
mL-SHADE[16]	10	10	10	10	10	10	5.04	1	2.16	10	**78.2**
GADE[18]	10	10	10	10	9.44	10	1	2	3	10	**75.44**
CMEAL[13]	10	10	10	10	10	10	1.4	1	1.04	10	**73.44**
HTPC[11]	10	10	10	10	10	10	0.28	0.04	3.04	10	**73.36**
UMDE-MS[6]	10	10	10	6.4	10	10	1	1	2	10	**70.4**
DLABC[17]	10	0	3.88	10	10	10	10	2	2	10	**67.88**
MiLSHADE-LSP[8]	10	10	10	2.92	10	5.2	0	0.6	2	10	**60.72**
ESP-SOMA[14]	9.68	0	4.12	10	4.08	10	1	1.04	2	10	**51.92**
AVERAGE	**9.9822**	**8.8888**	**9.3333**	**9.4067**	**9.64**	**9.7333**	**6.6267**	**3.2844**	**3.2178**	**10**	**80.1133**

†DISHchain1e+12 eventually achieved a score of 100, but the result was too late for this competition. See:
https://github.com/P-N-Suganthan/CEC2019/blob/master/Codes%20of%20DISHchain3e%2B12%20paper.zip

‡ Data were extracted from multiple tables to show the results when population sizes are tuned.

C. Secondary Algorithm Scores

Secondary algorithms faired significantly worse than primary algorithms, no doubt in part because they were not tuned. Table VIII tabulates secondary algorithm scores (averages limited to 4 decimal places). The reference number next to an algorithm's acronym refers *to the paper in which the contest result appears*, not the original reference for that algorithm. The best score among secondary algorithms was 75.72 (CIPDE), which would be 12th place if it were a primary algorithm. Also near the top of the list is a previous contest winner, the butterfly optimization/CMA hybrid algorithm EBOwithCMAR (74.32 and 73.04). The next seven algorithms are DE-based, with most being variations of the SHADE algorithm.

The ABC algorithm turned in the worst score (9.64) and could not find 10 digits for any function. The lbestPSO algorithm (11.72) and TWPSO algorithm (26.44) did not fare much better. CMA-ES (39.36) did well on the highly conditioned functions F1 and F2, but ultimately scored no better than the univariate approach, UNIVAR (40.2). Combining these last two methods

improved scores. In particular, the HS-ES method, which combines the CMA-ES with the univariate method, scored 50.32 and 47.6 as independently reported in two papers.

TABLE VIII.
SCORES FOR 18 SECONDARY ALGORITHMS.

Secondary Algorithm	Score by Function										Total Score
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	
CIPDE[9]	10	10	10	10	10	10	1	1.72	3	10	**75.72**
EBOwithCMAR[16]	10	10	10	10	10	10	0.56	1.68	2.08	10	**74.32**
EBOwithCMAR[12]	10	10	10	10	10	10	0.28	1.12	3	8.64	**73.04**
jDE	10	10	10	10	10	10	1.88	1.12	2.04	8	**73.04**
LSHADE_RSP[16]	10	10	10	6.76	10	10	1.36	1	2.2	10	**71.32**
JSO[16]	10	10	10	3.88	10	10	1	1.08	2.12	10	**68.08**
LSHADE_cnEpSin[16]	10	10	6.12	4.6	10	10	0.84	1.02	2.36	10	**64.94**
SHADE[15]	10	10	10	0	10	10	0	1	2	10	**63**
LSHADE[16]	10	10	7.16	0.24	10	10	1	1	2.04	10	**61.44**
ELSHADE-SPACMA[16]	10	10	5.44	0.84	10	10	1.08	1.16	3	7.88	**59.4**
HTPC[11]	10	10	5.68	1.72	10	10	0.04	0	3	6.76	**57.2**
HS-ES[12]	10	0.2	1.72	4.96	10	10	0.28	0.16	3	10	**50.32**
HS-ES[16]	7.6	0	1.72	4.96	10	10	0.16	0.16	3	10	**47.6**
iL-SHADE[8]	10	10	10	0	10	0	0	0	2.04	0	**42.04**
DFSABC_elite[17]	0	0	1.2	9.52	7	10	0.04	0.8	2	10	**40.56**
UNIVAR[15]	0	0	0	10	10	10	2	1	2	5.2	**40.2**
CMA-ES[11]	10	10	1.36	0	6.72	10	0	0	1.28	0	**39.36**
TW-PSO[11]	0	0	1	0	3.44	10	0	0	2	10	**26.44**
lbestPSO[11]	0	0	1.92	0	2.72	3.92	0	0	2	1.16	**11.72**
ABC[17]	0	0	1.36	0	5.16	1	0	0.08	2	0.04	**9.64**
AVERAGE	**7.38**	**6.51**	**5.734**	**4.374**	**8.752**	**8.746**	**0.576**	**0.705**	**2.308**	**7.384**	**52.469**

IV. DISCUSSION OF RESULTS

In the next section, we look at the top four algorithms to better understand the reasons for their success in this competition. Their top-tier performance is due, in part, to their success on the competition's most difficult functions, which are discussed in section *B*. Section *C* shows that in addition to their algorithmic strategies, their high function evaluation limits and large population sizes were a major factor in the top four algorithm's success. Section *D* looks at which algorithmic parameters competitors choose to tune.

A. Discussion of the Top Four Algorithms

jDE100 is derived, as its name suggests, from the jDE algorithm. Like jDE, jDE100 self-adapts the control parameters F and Cr that mediate DE's differential mutation and binomial crossover processes, respectively. These two parameters are adapted for each vector (i.e. F_i, Cr_i) based on feedback from function values. jDE100 also introduces several new features. Unlike its predecessor, jDE100 maintains both big and small populations. The primary purpose of the small population is to archive copies of the best vectors found by the big population. When the big population is evolving, the small population serves as a source for

the lead vector in the differential mutation equation, with the remaining two difference vectors being drawn from the big population. When the small population is evolving, all three vectors in the differential mutation equation are drawn from the small population. In addition, jDE100 sets upper and lower limits for F_i and Cr_i differently compared to jDE. It is these lower limits that were tuned.

The jDE100 algorithm also includes a restart mechanism to manage population diversity. In particular, populations are reinitialized when the percentage of better vectors with similar function values, i.e. function values that differ by 1.0e−16 or less, exceeds a given amount (*myEqs*). In addition, the big population is reinitialized when there is no improvement in the best vector over 1.0e+9 function evaluations. The best vector in the small population is left unaltered by initialization. Except for choosing the component vectors differently and making *F* and *Cr* self-adaptive, jDE100 relied on the classic DE strategy: DE/rand/1/bin. Survivor selection was also the same as traditional DE algorithms. Also similar to classic DE, population sizes are not dependent on the number of function evaluations. (The paper with correctly tuned parameter values is online with the jDE100 code package).

DISHchain1e+12 is, as its name suggests, a version of the DISH algorithm, which itself is an extension of L-SHADE. Tuning the population size and extending function evaluation limits to 1.0e+12 appear to be what distinguishes DISHchain1e+12 from DISH. In the DISH algorithm, both F_i and Cr_i are self-adaptive, but the feedback for the adaptation process comes from a distance measure; specifically, adaptation weights are computed from the Euclidean distance between the target and trial vectors. Both F_i and Cr_i are also modified based on the ratio of the current number of function evaluations to the maximum allowed number of function evaluations, or FEs ratio. Like jSO, the DISHchain1e+12 algorithm relies on the current-to-*p*best-w/1 mutation strategy:

$$\mathbf{v}_i = \mathbf{x}_i + F_{w,i}(\mathbf{x}_{p\text{Best}} - \mathbf{x}_i) + F_i(\mathbf{x}_{r1} - \mathbf{x}_{r2}),$$

where $\mathbf{x}_{p\text{Best}}$ is one of the better p vectors in the population. The value of p is designed to increase linearly with the FEs ratio. Both the scale factor and the weighting factor $F_{w,i}$ as well as the crossover parameter are also determined in part by the FEs ratio. In addition, $\mathbf{x}_{r1}$ is a randomly chosen solution from the population, but $\mathbf{x}_{r2}$ is a randomly chosen solution from the union of the population and an external archive. Except for *Cr* being adaptive, DISHchain1e+12 relies on binomial crossover and DE's traditional selection criterion. As in L-SHADE, the population size is also dependent on the FEs ratio. DISHchain1e+12 does not invoke a restart mechanism in case of stagnation. Programming note: In line 118 of main.cc, reset NP_0 from 25 to 250, 2500 and 10,000 for F4, F7 and F8, respectively.

HyDE-DF also self-adapts both *F* and *Cr* in the same way as jDE, i.e. based on feedback from function values. It employs a combined line-recombination/differential mutation operation known as DE/target-to-perturbed best/1 (subscript *G* is suppressed here):

$$\mathbf{v}_i = \mathbf{x}_i + \delta \cdot F^1{}_i(\varepsilon \cdot \mathbf{x}_{\text{best}} - \mathbf{x}_i) + F^2{}_i(\mathbf{x}_{r1} - \mathbf{x}_{r2}).$$

The perturbation factor ε is drawn from a normal distribution whose standard deviation is 1.0 and whose mean is a third scale factor, F^3. The decay factor δ, gradually decreases from 1 to 0 as the number of function evaluations approach *DFt* generations. The effect of the decay factor is to slow convergence toward the population's best vector. If there is no improvement in 1.0e+4 generations, the population is reinitialized around a group of the 10 best vectors. HyDE-DF relies on classic DE survivor selection and binomial crossover (albeit with adaptive *Cr*). Only the population size, which is constant throughout a trial, was tuned for this competition.

SOMA-T3A was the only one of the top 4 algorithms that was not DE-based. SOMA's basic strategy is a line-search from $\mathbf{x}_i$ (a "migrant") to $\mathbf{x}_L$ (the "leader"), except that not every component of the difference term ($\mathbf{x}_L - \mathbf{x}_i$) is applied. We have taken some liberties with notation to rewrite the basic SOMA generating equation here so that it can be more easily compared to the other winning strategies. As in the other two equations above, the generation index has been suppressed and $\mathbf{v}_i$ represents a trial vector. With that understanding, the core generating equation for SOMA algorithms becomes:

$$\mathbf{v}_i = \mathbf{x}_i + t \cdot (\mathbf{x}_L - \mathbf{x}_i) \cdot \mathbf{p}_i.$$

The *PRTVector* $\mathbf{p}_i$ is a vector of length D whose components are either 0 or 1. When $\mathbf{p}_i = [1, 1\ldots, 1]$, the above equation reduces to a line-search that starts at $\mathbf{x}_i$ ($t = 0$) and moves toward the leader $\mathbf{x}_L$ in discrete steps of size *Step* until $t \geq PathLength$. In SOMA-T3A, however, individuals jump in discrete steps toward the leader until they reach a given maximum number of jumps (*Njumps*),

so that *PathLength* = *Step*·*Njumps*. New values for each component $p_{i,j}$ of $\mathbf{p}_i$ are generated before each jump by comparing a uniformly distributed random number $0 < rand_j \leq 1$ to the perturbation parameter $0 < prt_i \leq 1.0$. If $rand_j < prt_i$, $p_{i,j} = 1$; otherwise, $p_{i,j} = 0$. As a result, the trial point only moves in directions j whose corresponding component $p_{i,j}$ is 1. Multiplication by the *PRTVector* has the same effect as binomial crossover between $\mathbf{x}_i$ and the line-recombinant vector $\mathbf{x}_i + t\cdot(\mathbf{x}_L - \mathbf{x}_i)$, with prt_i playing a role similar to that of *Cr* in DE: components for $\mathbf{v}_i$ come from $\mathbf{x}_i$ when $rand_j \geq prt_i$ and from $\mathbf{x}_i + t\cdot(\mathbf{x}_L - \mathbf{x}_i)$ when $rand_j < prt_i$. The best solution $\mathbf{v}_i^*$ along the path replaces $\mathbf{x}_i$ if $f(\mathbf{v}_i^*) \leq f(\mathbf{x}_i)$.

The "A" in T3A refers to the adaptive processes that determine prt_i and *Step*. Starting with a value of 0.08, the perturbation parameter prt_i increases to 0.98 as the number of function evaluations reaches a preset maximum. Similarly, *Step* also depends on the current number of function evaluations (*FE*s), i.e. $Step = 0.02 + 0.005\cdot\cos(0.5\pi 10^{-7} FEs)$. The "T3" in T3A stands for the Team-to-Team strategy, i.e. how migrants and leaders are chosen. Two subpopulations sizes m and k are randomly selected from the population. Based on function value, the best n of m individuals become migrants and the single best of k individuals is designated as the leader. Each migrant then moves towards the leader as described above. Both m and k were tuned for this competition, while n was constant and equal to 4.

In summary, three of the four winning algorithms were remarkably similar. Each was a DE-based algorithms that relied on both binomial crossover and DE's traditional greedy selection criterion. Similarly, each of these three algorithms made both *F* and *Cr* self-adaptive. Both jDE100 and HyDE-DF adapted parameters based on function value like jDE, while DISHchain1e+12 evolved adaptation weights with a distance-based measure. None of these three algorithms was memetic, with each relying instead on just one strategy. More specifically, the DISHchain1e+12 and HyDE-DF algorithms employed similar mutation/recombination operators (see above), whereas jDE100's mutation operation is just DE/rand/1 with a different way of selecting the component vectors.

Although it does not rely on differential mutation, SOMA-T3A nevertheless shares several features with the other three top-tier algorithms. In particular, SOMA-T3A employs both a greedy selection criterion and a binomial crossover-like operation. SOMA-T3A also adapts parameters (albeit different from *F* and *Cr*) and eschews the memetic approach. Similarly, SOMA's generating equation and population structures take advantage of the population's better individuals, as do the other top-tier algorithms.

Given the similarities in these algorithms, it would appear that the combination of low limits for *F*, holding $Cr = 1$ and perhaps restarts are the strategies that helped jDE100 do so well on F9. Large population sizes and correspondingly high function evaluation limits were also a factor in the success of both jDE100 and DISHchain1e+12 on F9. On F8, the success of the top four algorithms is more likely due to their high limits on function evaluations alone. Finally, their reliance on the population's better vectors does not appear to have significantly affected the robustness of these algorithms.

B. Discussion of Function Difficulty

Table IX ranks the functions from easy (1) to hard (10) based on the average number of correct digits found by the 18 primary, 20 secondary and 38 combined algorithms. Averages are limited to 4 decimal places. Three functions stand out as being particularly difficult. Among the primary algorithms, the Happy Cat function (F9) was the hardest (3.2178 correct digits on average), with the Expanded Schaffer function (F8) coming in a close second (3.2844 correct digits) and the modified Schwefel function (F7) coming in a distant third (6.2667 correct digits). Even though the average number of digits found by primary algorithms for function F8 and F9 are very close, Table VII shows that the top four primary algorithms found all 10 digits for F8, but only jDE100 scored 10 on F9, making the Happy Cat function the most difficult function for the primary algorithms by either measure. By contrast, the modified Schwefel function (F7) (0.576 digits) was the most difficult for secondary algorithms, followed by the Expanded Shaffer function (0.705 digits) and Happy Cat (2.308 digits).

C. Discussion of the Effect of Function Evaluation Limits

Table X compares the average number of function evaluations taken by jDE100 to find all 10 digits for each function with the $D\cdot$1.0e+4 limit chosen for a recent CEC contest on real parameter optimization. In only one case, F6, would the CEC limit be high enough for jDE100 to find all ten digits. Table XI shows the limits on the maximum allowed number of function evaluations chosen by the 18 primary algorithms for this competition. The limits chosen by contestants were always higher than the prior CEC limit of $D\cdot 10^4$ except in two cases where they were equal to the CEC limits: the DISH algorithm on F1 and DISHchain1e+12 on F6. The moral is: limits set as low as they were for prior CEC competitions reward the Hare.

As mentioned above, fully solving the Expanded Schaffer function put four algorithms ahead of the rest. In large part, the strategy for solving F8 is the same as that of Tortoise: persistence and patience, i.e. allowing for a very large number of function evaluations. For example, Table XIII shows that solving F8 took approximately 1.219e+9, 8.0e+10, 1.6e+9 and 7.0e+8 function evaluations for jDE100, DISHchain1e+12, HyDE-DF and SOMA-T3A, respectively. Except for HyDE-DF, these evaluation

TABLE IX.
FUNCTION DIFFICULTY RANKED BY THE AVERAGE NUMBER OF DIGITS FOUND

Function	Primary (18)		Secondary (20)		Combined (38)	
	Rank	Average	Rank	Average	Rank	Average
Chebyshev	2	9.9822	4	7.38	4	8.6126
Inverse Hilbert	7	8.8889	5	6.51	5	7.6368
Lennard-Jones	6	9.333	6	5.734	6	7.4389
Rastrigin	5	9.4067	7	4.374	7	6.7578
Griewangk	4	9.64	1	8.752	2	9.1726
Weierstrass	3	9.7333	2	8.746	1	9.2136
Modified Schwefel	8	6.2667	10	0.576	8	3.4421
Expanded Schaffer	9	3.2844	9	0.705	10	1.9268
Happy Cat	10	3.2178	8	2.308	9	2.7389
Ackley	1	10	3	7.384	3	8.6231
Average Score:		**80.1132**		**52.469**		**65.5632**

times correlate reasonably well with the population sizes if one computes the ratio of the logarithm of the number of function evaluations to the logarithm of the population size (Table XIII).

As was the case for F8, the key to finding the solution to F9 was patience, a substantial population and a suitable search strategy. For F9, jDE100 needed almost as many function evaluations as it did for F8. Algorithms that did not score well on F8 and F9 should repeat their experiments with longer run times and perhaps larger populations than they did for this contest. If a solution is still not possible, then one must suspect that the algorithmic strategy is to blame.

TABLE X.

FUNCTION EVALUATIONS TO FIND 10 CORRECT DIGITS BY JDE100 VS. PRIOR CEC EVALUATION LIMITS

Function	jDE100 (avg.)	CEC: $D \cdot 10^4$	FE_{jDE100} / FE_{CEC}
F1	1.947e+05	9e+4	2.163
F2	2.332e+06	1.6e+5	14.58
F3	8.854e+05	1.8e+5	4.919
F4	3.998e+05	1.0e+5	3.998
F5	1.851e+05	1.0e+5	1.851
F6	3.497e+04	1.0e+5	0.3497
F7	1.127e+07	1.0e+5	112.7
F8	6.444e+08	1.0e+5	6,444
F9	9.353e+08	1.0e+5	9,353
F10	9.622e+05	1.0e+5	9.622

TABLE XI
TERMINATION CONDITION WHEN 10 DIGITS NOT FOUND

Algorithm	Maximum Allowed Function Evaluations
UMDE-MS	*D*·3.0e+4
DISH	9.0e+4 → 2.0e+7, depending on function (tuned)
MiLSHADE-LSP	1.0e+5 → 5.0e+5, depending on function (tuned)
rCIPDE	3.0e+8
jDE100	1.0e+12 (not needed)
rjDE	1.0e+7
HTPC	3.0e+6
ESHADE-USM	3.5e+6
CMEAL	1.0e+6 → 1.0e+9, depending on function (tuned)
ESP-SOMA	1.0e+7
Co-Op	5.0e+6 (SHADE), 1.0e+6 (Univariate Sampling), 1.0e+5 (CPSO)
mL-SHADE	2.0e+6
DLABC	No improvement to best solution in 1.0e+4 function evaluations
GADE	1.15e+8
HyDE-DF	*Np*·8.0e+6. F1–F6, F10: 4.0e+8; F7, F9: 8.0e+8; F8: 1.6e+9
SOMA Pareto	6.0e+8
SOMA T3A	1.0e+9
DISHchain1e+12	1.0e+5 → 1.0e+12, depending on function (tuned)

TABLE XII
RESOURCES NEEDED BY THE TOP FOUR ALGORITHMS TO OPTIMIZE F8

Algorithm	jDE100	DISHchain1e+12	HyDE-DF	SOMA-T3A
Function Evaluations	9.353e+8 (avg.)	8.0e+10 (est.)	8.0e+8 (est.)	7.0e+8
Population Size	1000	31,623	200	1500
log(FEs)/log(Pop)	2.99	2.42	3.87	2.78

D. Discussion of Tuning

An unusual feature of the 100-Digit Challenge is that it permits tuning. Table XIII show which parameters were tuned by which algorithms. Surprisingly, several primary algorithms (rCIPDE, HTPS and Co-Op) were not tuned, except perhaps to give good performance across the whole testbed. Given that not all functions had the same dimension, we expected that most entrants would opt to tune the initial population size, but only four did (DISH, ESP-SOMA, DLABC and HyDE-DF). The CMEAL algorithm chose to tune the minimum population size and SOMA T3A tuned the sizes of two auxiliary populations.

Even though it was unconstrained, more than one contestant chose to tune the maximum allowed number of function evaluations. This approach was due in large part to the method of decreasing the population size as the search nears the limit on function evaluations and more generally due to the dependence of parameters on the ratio of the number of function evaluations-so-far to the maximum allowed number of function evaluation (as in the L-SHADE algorithm). Making an algorithmic parameter dependent on an unknown was a problem in the case of F8 and F9, where the average number of evaluations in the winning entry was more than 6,000 and 9,000 times greater, respectively, than what was allowed in prior CEC competitions. While it must be admitted that jDE100's large population inflated its run times, even HyDE-DF's solution to F8 with a smaller population took far longer than the limits set on the maximum number of function evaluations that some had allowed.

For the record, we note that jDE's original "final" entry (i.e. the one that appears in the proceedings) very slightly over-tuned their algorithm by adjusting the minimum values for *Cr* and two scale factors, one for the big population and one for the small population. Upon being notified of this oversight, Team jDE100 immediately reran their algorithm in accordance with the rules. Their solution was to tune just a single scale factor for both populations. They still posted a score of 100.

TABLE XIII
TUNED PARAMETERS DESCRIBED

Algorithm	Parameter	Description
UMDE-MS	Gp_1	#generations without improvement before calling univariate method, FE ≤ 0.8 Max_FE
	Gp_2	#generations without improvement before calling univariate method, FE > 0.8 Max_FE
DISH	*MAXFES*	Maximum allowed number of function evaluations
	NP_{init}	Initial population size
MiLSHADE-LSP	*maxevals*	Maximum allowed number of function evaluations
	I_{Step}	initial step-size (as a percentage) in Solis-Wets local search method
rCIPDE	-	No tuning
jDE100	F_l	Lower limit for scale factor *F*
	CR_l	Lower limit for crossover parameter *Cr*
rjDE	*NP*	Population size
HTPC	-	No Tuning
ESHADE	*cr*	Crossover parameter; only tuned for F7 and F8; adaptive otherwise.
CMEAL	*maxFES*	Maximum allowed number of function evaluations
	*N*min	The final value of population size at the end of the search process if *maxFES* is reached
ESP-SOMA	*NP*	Population size
	adaptivePRT	Boolean parameter controlling crossover
Co-Op	-	No tuning
mL-SHADE	N^{stuck}	The maximum number of generations without a memory update
DLABC	*SN*	"number of food sources"; population size
	α	A real number (0,1) to control the mutation rate (crossover probability)
GADE	GA maxGen	Maximum number of generations for GA phase
	ε_d	"duplicate epsilon"; tidal mutation parameter
HyDE-DF	*NP*	Population size
SOMA T3A	m	Population size from which *n* migrants are chosen
	k	Population size from which the best is chosen to be the Leader
SOMA Pareto	*PRT*	Perturbation probability
DISHchain1e+12	*MAX_FES*	Maximum allowed number of function evaluations
	NP_0	Initial population size. Not scriptable. Set NP_0 in line 118 of main.cc for F4, F7 and F8.

V. Conclusions

One of the goals of this competition was to show that time-limited competitions rarely allow enough time for global optimization algorithms to do their work. For example, the winning entry would have been able to optimize only one function (F6) to ten digits of accuracy in the time allowed by a prior CEC competition. Granted, the total population size chosen by jDE100 was quite large (1000) and not tuned for each function. Hence, run times for jDE100 on the easier functions are unnecessarily high (plots in the Appendix show this), but certainly the number of function evaluations needed to optimize F8 and F9 was far in excess of what would have previously been allowed. These results illustrate that to discover an algorithm's problem solving potential, we need to treat run time as a measurement, not a constraint.

Only three of this year's functions posed a significant challenge to most algorithms. It is clear that even when rotated and shifted, some of the older functions like Ackley, Weierstrass, Griewangk and to a lesser extent, Rastrigin, are no longer a challenge. We can solve them. Increasing their dimension beyond 10 does make them harder (except for Griewangk, which gets *easier* after about $D = 12$), but not intractable. We are *way* past the era of optimizing the sphere, cigar, tablet and rotated ellipse functions and are now reaching beyond the stable of functions that were once thought to be exceptionally difficult. To more ably determine an algorithm's effectiveness, we will need a new and more demanding set of test functions, especially ones that are difficult for DE-based methods. There is a risk, however, that such functions will be difficult by design, like F9, without capturing the difficulty of real-world problems, as do F1, F2 and F3.

One of the surprise takeaways from this year's contest was the inability of memetic algorithms to compete with single strategy methods. Granted, results are limited, but clearly the lack of a memetic approach was not a handicap. It did seem, however, that the addition of a univariate sampling method to CMA-ES was mutually beneficial, with HS-ES outperforming either algorithm alone.

As in other competitions, algorithms with self-adaptive control parameters generally did well, showing that tuning certain control parameters can be automated without impairing the algorithm's performance. The effectiveness of adapting the population size based on the FEs ratio was less in evidence, probably because the maximum allowed number of generations typically was too low for both F8 and F9. For example, SOMA-T3A found the solution to F8 in 7.0e+8 function evaluations, but primary algorithms ESHADE-USM, ML-SHADE and MiLSHADE-LSP set the limit at 3.5e+6, 2.0e+6 and 5.0e+5, respectively. In addition, the long stagnation periods for F8 and F9 are a cautionary tale for anyone halting trials after a failure to improve after a preset number of generations.

DE-based algorithms dominated the competition both in number and in scoring. It is interesting to note that despite their significant differences, most relied on traditional DE binomial crossover, in part because many were derived from the SHADE algorithm. Of course, unlike traditional DE crossover, SHADE variants adapt the crossover parameter, Cr. Not only did SHADE variants adopt binomial crossover, they, along with the SOMA and DLABC algorithms, also shared the same survivor selection criterion as DE. Perhaps surprisingly, the top performing algorithm in this competition also relied on the basic DE mutation strategy: DE/rand/1, albeit with an adaptive value for the mutation scale factor F and different rules for selecting the component vectors. The other two top contender's search strategies were modification of another early DE recombination/mutation strategy: DE/current-to-best/1. While DE-based algorithms dominated, SOMA T3A also did well.

We had hoped that the comparatively low "success" rate of 50% and the freedom to tune algorithmic parameters would have made population sizes more representative of what is appropriate in practice. Team jDE100, however, saw a benefit to using a single, large population (~1000) which they did not have to tune because the number of allowed number of function evaluations was unlimited. This then gave them the freedom to tune two alternative parameters (lower limits for F and Cr) although it did inflate run times for the easier functions. Clever!

In closing, we would like to thank everyone who participated in this year's first ever 100-Digit Challenge on real parameter, single objective optimization. We were happy to receive so many submissions and gratified that no one tried any "tricks". Well done!

Acknowledgments

We would like to express our gratitude to GECCO competition chair, Dr. Markus Wagner, for allowing us both to offer our challenge at GECCO2019 and to combine the results with those of CEC2019. We would also like to extend our gratitude to CEC2019 competition chair, Dr. Jialin Liu, for embracing this new competition. Finally, we would like to thank the reviewers who provided many insightful comments.

REFERENCES

[1] D. Mitra, F. Romeo and A.S. Vincentelli, "Convergence and Finite-Time Behavior of Simulated Annealing," *Advances in Applied Probability*, Sept. 1986.

[2] N. Trefethen, "A Hundred-dollar, Hundred-digit Challenge," *SIAM News*. 35 (1): 65, 2002.

[3] F. Bornemann, D. Laurie, S. Wagon, J. Waldvogel, *The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2004.

[4] K. V. Price, N. H. Awad, M. Z. Ali, and P. N. Suganthan, "The 100-Digit Challenge: Problem Definitions and Evaluation Criteria for the 100-Digit Challenge Special Session and Competition on Single Objective Numerical Optimization," Technical Report, Nanyang Technological University, Singapore, November 2018. https://github.com/P-N-Suganthan

[5] H.G. Beyer, S. Finck, "HappyCat – A simple function class where well-known direct search algorithms do fail," in C.A.C Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia and M. Pavone, (eds) *Parallel Problem Solving from Nature - PPSN XII*. PPSN 2012. *Lecture Notes in Computer Science*, vol. 7491. Springer, Berlin, Heidelberg.

[6] Y. Fu and H. Wang, "A univariate marginal distribution resampling differential evolution algorithm with multi-mutation strategy," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 1236–1242.

[7] A. Viktorin, R. Senkerik, M. Pluhacek, T. Kadavy and A. Zamuda, "DISH algorithm solving the CEC 2019 100-digit challenge," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 1–6.

[8] D. Molina and F. Herrera, "Applying memetic algorithm with improved L-SHADE and local search pool for the 100-digit challenge on single objective numerical optimization," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 7–13.

[9] S.X. Zhang, W.S. Chan, K.S. Tang and S.Y. Zheng, "Restart based collective information powered differential evolution for solving the 100-digit challenge on single objective numerical optimization," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 14–18.

[10] J. Brest, M.S. Maučec, B. Bŏsković, "The 100-digit challenge: algorithm jDE100," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 19–26.

[11] P. Xu, W. Luo, X. Lin, Y. Qiao and T. Zhu, "Hybrid of PSO and CMA-ES for global optimization," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 27–33.

[12] A. Kumar, R. K. Misra, D. Singh, and S. Das, "Testing a multi-operator based differential evolution algorithm on the 100-digit challenge for single objective numerical optimization," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 34–40.

[13] P. Bujok and A. Zamuda, "Cooperative model of evolutionary algorithms applied to CEC 2019 single objective numerical optimization," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, DOI:10.1109/CEC.2019.8790317.

[14] T. Kadavy, M. Pluhacek, R. Senkerik and A. Viktorin, "The ensemble of strategies and perturbation parameter in self-organizing migrating algorithm solving CEC 2019 100-digit challenge," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, DOI:10.1109/CEC.2019.8790012.

[15] G. Zhang, Y. Shi and J.S. Huang, "Cooperative optimization algorithm for the 100-digit challenge," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, DOI:10.1109/CEC.2019.8790272.

[16] J.F. Yeh, T.Y. Chen and T.C. Chiang, "Modified L-SHADE for single objective real-parameter optimization," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, DOI:10.1109/CEC.2019.8789991.

[17] J. Lu, X. Zhou, Y. Ma, M. Wang, J. Wan and W. Wang, "A novel artificial bee colony algorithm with division of labor for solving CEC 2019 100-digit challenge benchmark problems," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, DOI:10.1109/CEC.2019.8790252.

[18] A. Epstein, M. Ergezer, I. Marshall and W. Shue, "GADE with fitness-based opposition and tidal mutation for solving IEEE CEC2019 100-digit challenge," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, June 10–13, Wellington, New Zealand, 2019, pp. 395–402.

[19] F. Lezama, J. Soares, R. Faia, and Z. Vale, "Hybrid-adaptive differential evolution with decay function (HyDE-DF) applied to the 100-digit challenge competition on single objective numerical optimization," in *GECCO Companion*, July 13–17, Prague, Czech Republic, 2019, p. 6. ACM ISBN 978-1-4503-6748-6/19/07. https://doi.org/10.1145/3319619.3326747.

[20] Q. B. Diep, I. Zelinka and S. Das, "Self-organizing migrating algorithm for the 100-digit challenge," in GECCO Companion, July 13–17, Prague, Czech Republic, 2019, p. 3. ACM ISBN 978-1-4503-6748-6/19/07. https://doi.org/10.1145/3319619.3326750.

[21] A. Zamuda, "Function evaluations up to 1e+12 and large population sizes assessed in distance-based success history differential evolution for 100-digit challenge and numerical optimization scenarios (DISHchain1e+12)," in *GECCO Companion*, July 13–17, Prague, Czech Republic, 2019, p. 11. ACM ISBN 978-1-4503-6748-6/19/07. https://doi.org/10.1145/3319619.3326751.

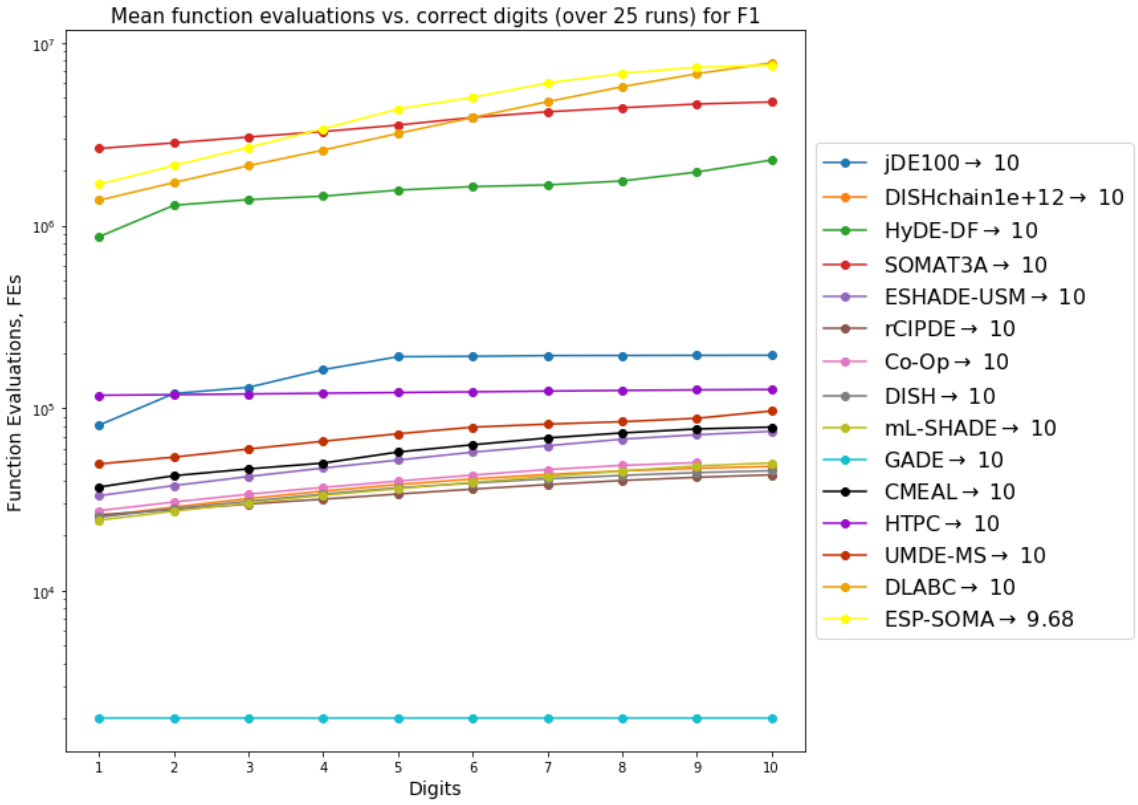
[22] C. T. Thanh, Q. B. Diep, I. Zelinka, and R. Senkerik, "Pareto-based Self-Organizing Migrating Algorithm solving 100-Digit Challenge," in *Proceedings of the 2019 Swarm, Evolutionary and Memetic Computing Conference*, July 10–12, Maribor, Slovenia, EU, 2019, pp. xx-xx.

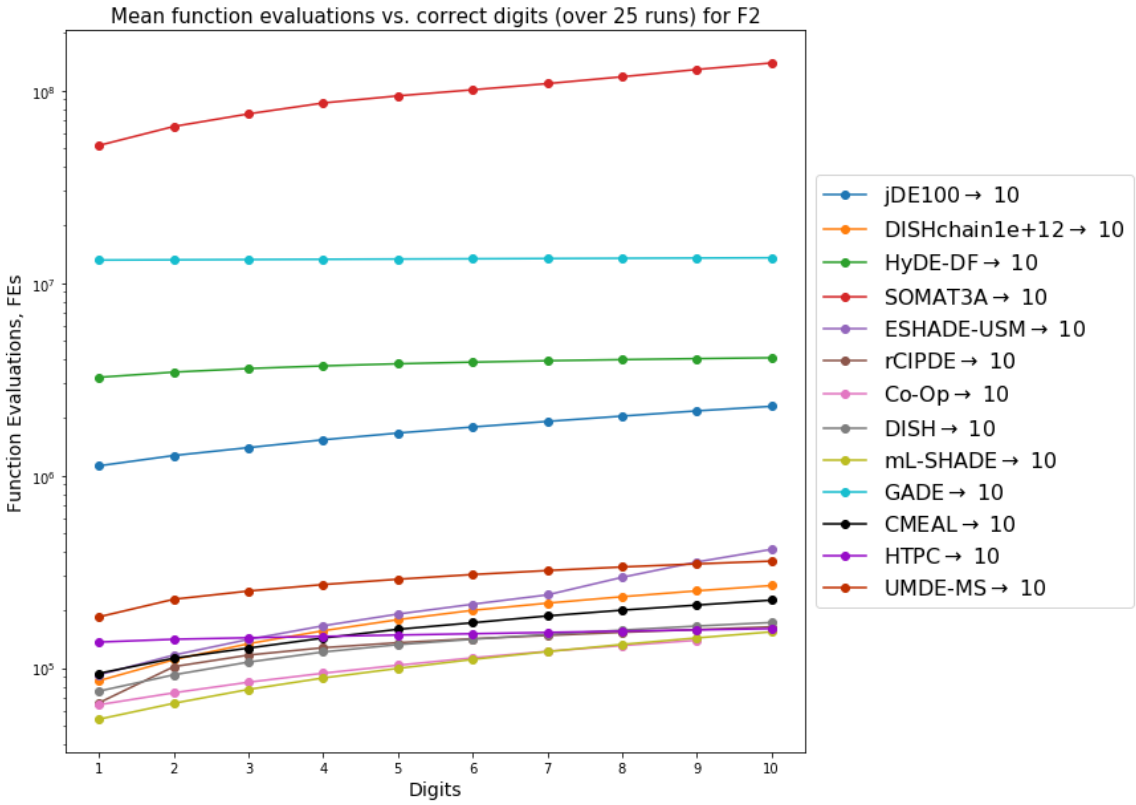
[23] Q. B. Diep, I. Zelinka, S. Das and R. Senkerik, "SOMA T3A for Solving the 100-Digit Challenge," in *Proceedings of the 2019 Swarm, Evolutionary and Memetic Computing Conference*, July 10–12, Maribor, Slovenia, EU, 2019, pp. xx–xx.

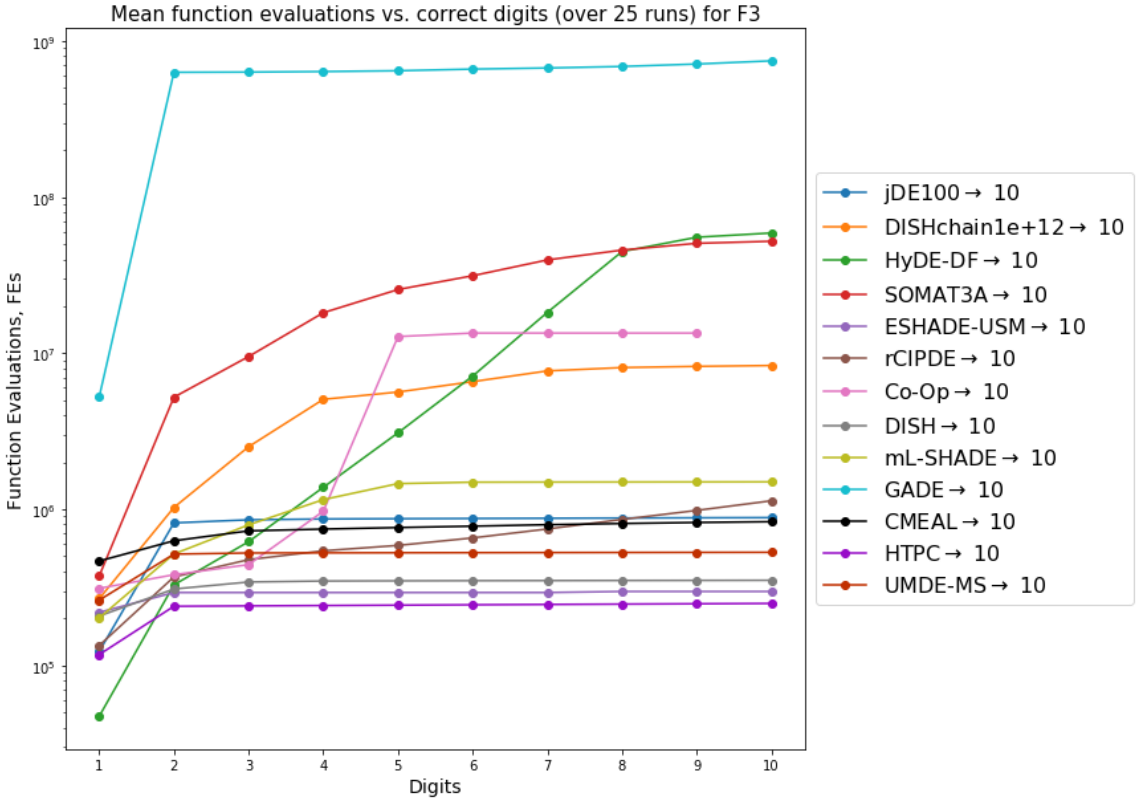
[24] A. Alić, K. Berkovič, B. Bŏsković and J. Brest, "Population Size in Differential Evolution," in *Proceedings of the 2019 Swarm, Evolutionary and Memetic Computing Conference*, July 10–12, Maribor, Slovenia, EU, 2019, pp. xx–xx.

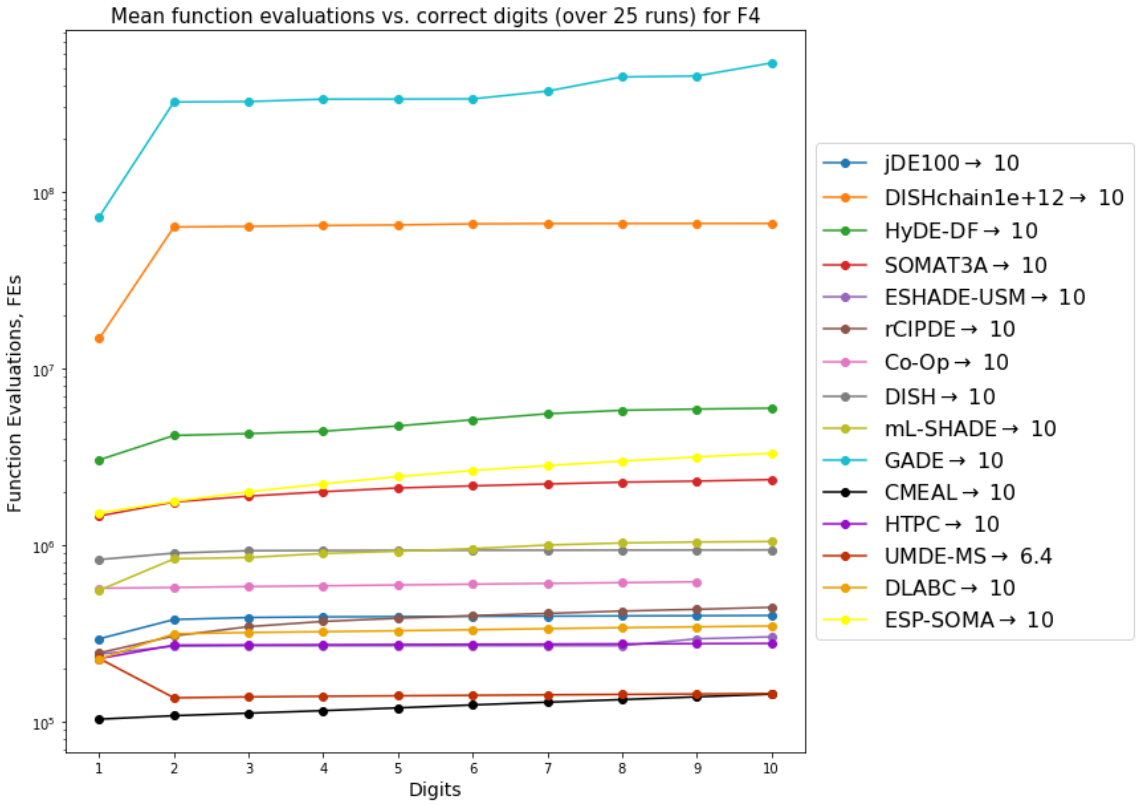
APPENDIX (CEC AND GECCO RESULTS ONLY)

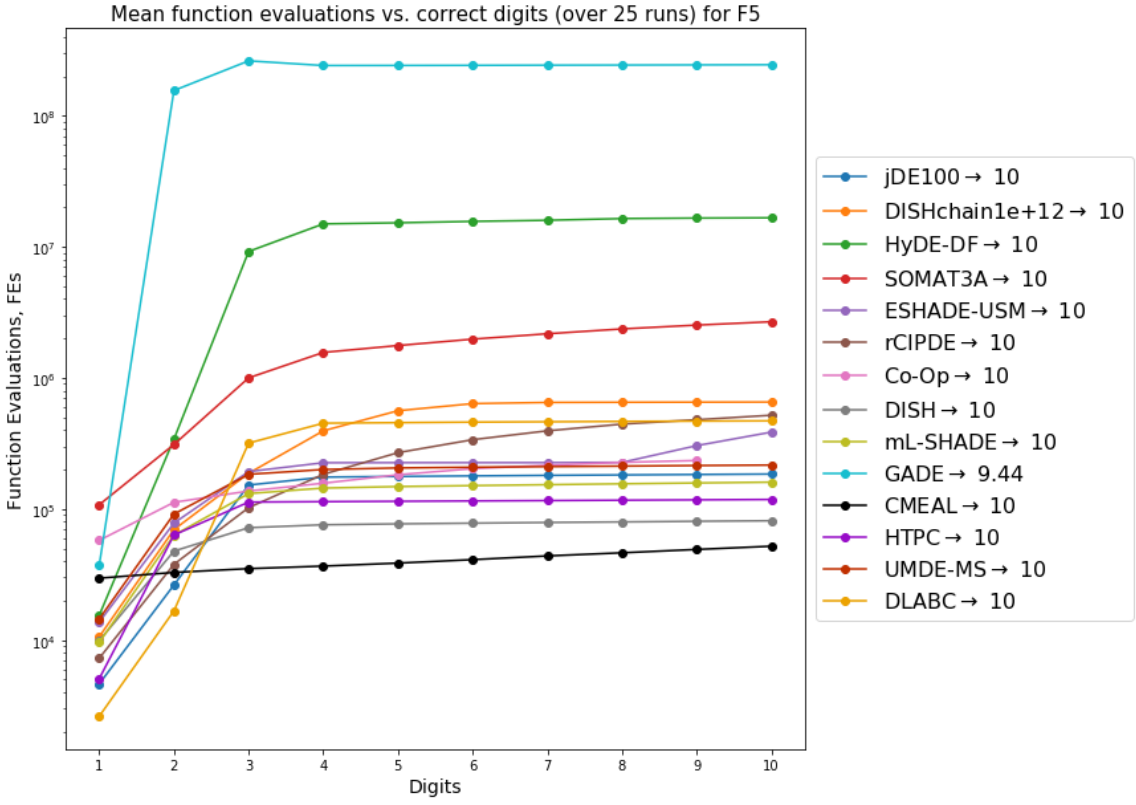
Below are 10 graphs (one for each function) that plot the average number of function evaluations taken to find each of 10 digits. Only cases in which all 25 trials found all ten digits are plotted. Numbers to the right of each algorithm's acronym in the legend is the score for that algorithm-function combination.

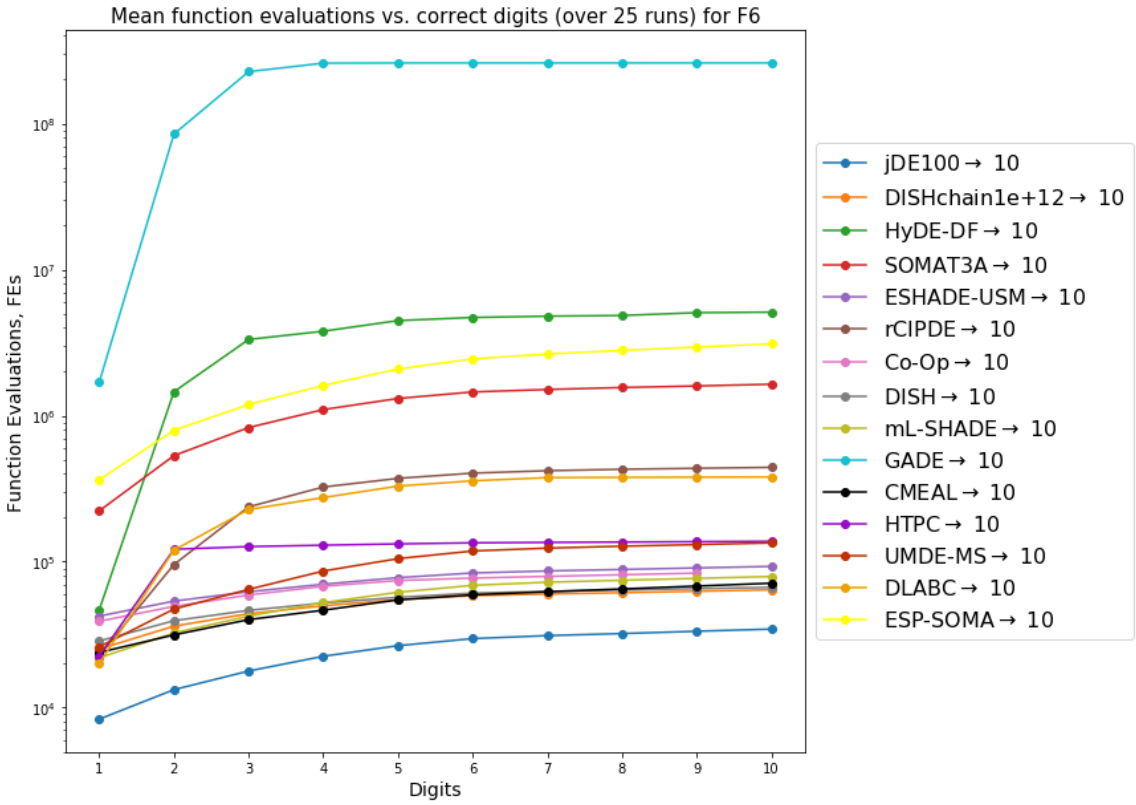


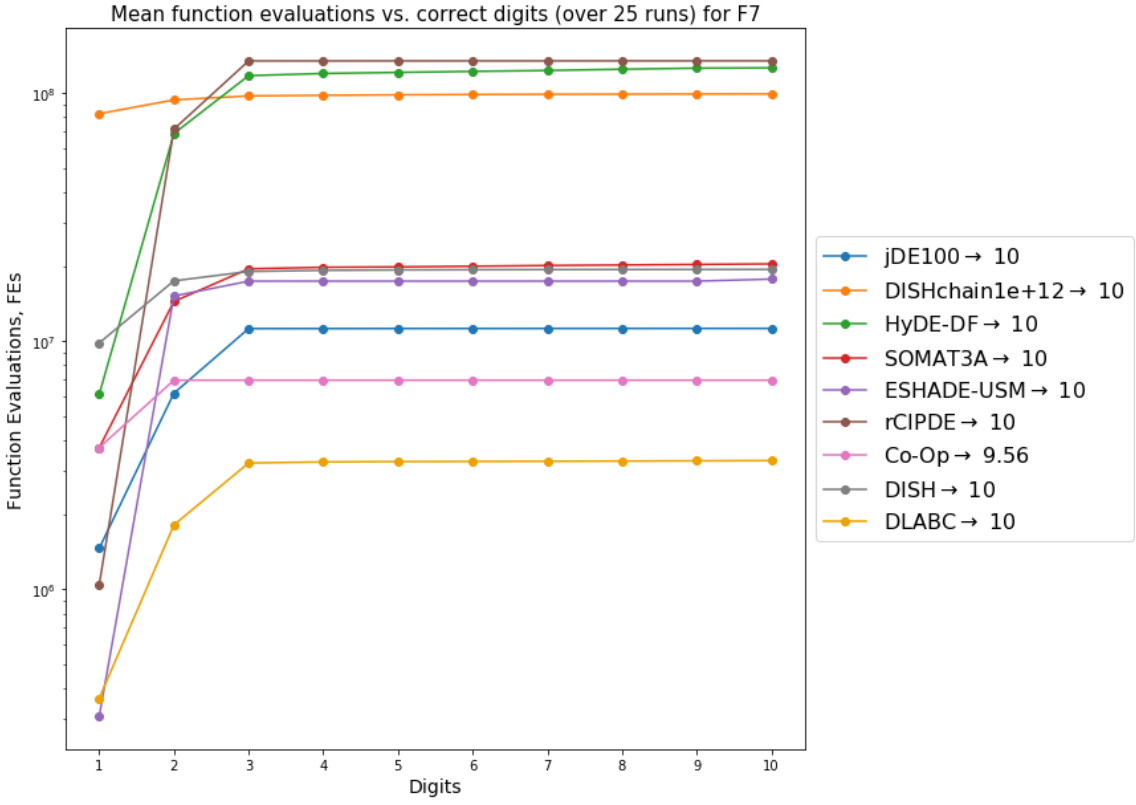

Mean function evaluations vs. correct digits (over 25 runs) for F2
Function Evaluations, FEs
Digits
jDE100→ 10
DISHchain1e+12→ 10
HyDE-DF→ 10
SOMAT3A→ 10
ESHADE-USM→ 10
rCIPDE→ 10
Co-Op→ 10
DISH→ 10
mL-SHADE→ 10
GADE→ 10
CMEAL→ 10
HTPC→ 10
UMDE-MS→ 10

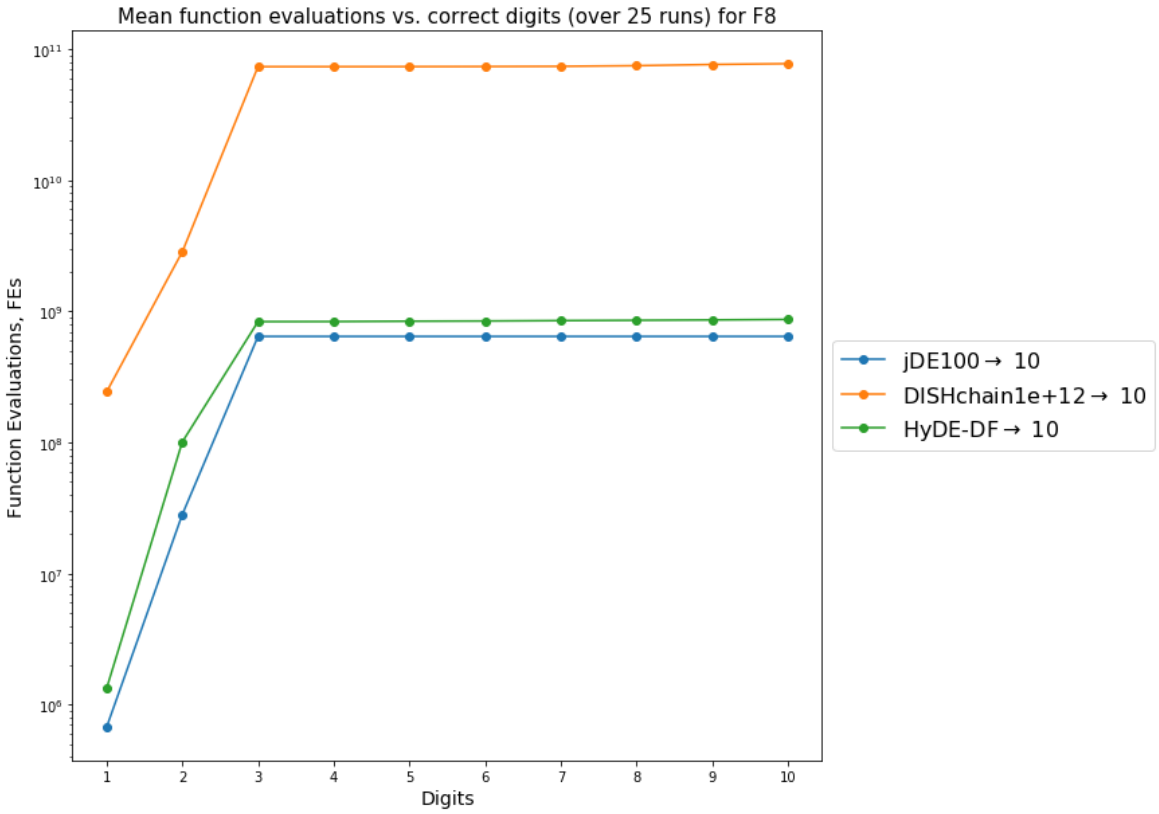

Mean function evaluations vs. correct digits (over 25 runs) for F3
Function Evaluations, FEs
Digits
jDE100→ 10
DISHchain1e+12→ 10
HyDE-DF→ 10
SOMAT3A→ 10
ESHADE-USM→ 10
rCIPDE→ 10
Co-Op→ 10
DISH→ 10
mL-SHADE→ 10
GADE→ 10
CMEAL→ 10
HTPC→ 10
UMDE-MS→ 10

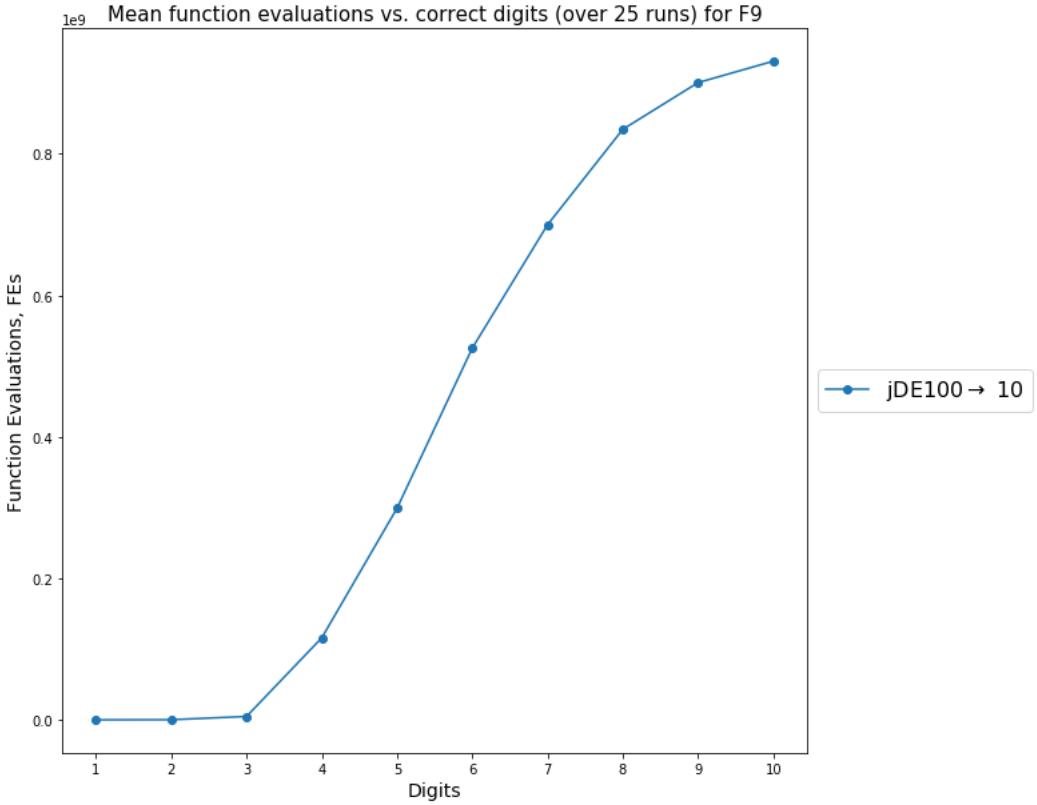

Mean function evaluations vs. correct digits (over 25 runs) for F4
Function Evaluations, FEs
Digits
jDE100→ 10
DISHchain1e+12→ 10
HyDE-DF→ 10
SOMAT3A→ 10
ESHADE-USM→ 10
rCIPDE→ 10
Co-Op→ 10
DISH→ 10
mL-SHADE→ 10
GADE→ 10
CMEAL→ 10
HTPC→ 10
UMDE-MS→ 6.4
DLABC→ 10
ESP-SOMA→ 10

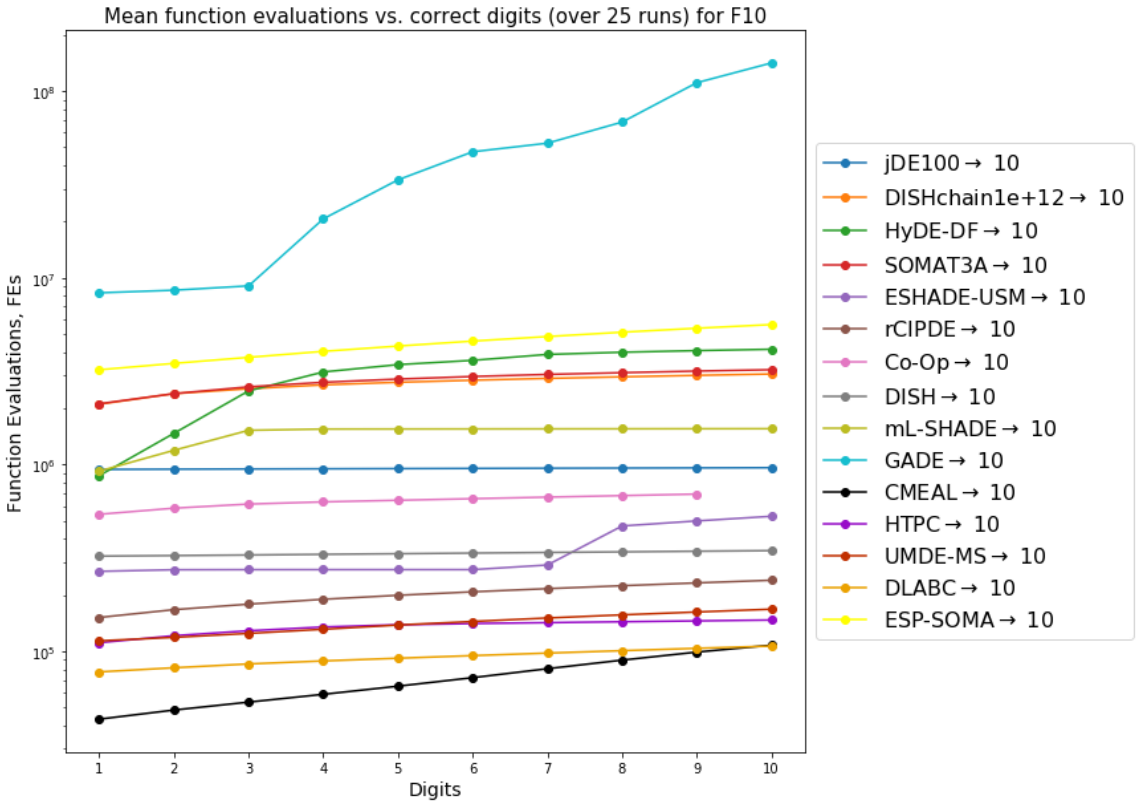

Mean function evaluations vs. correct digits (over 25 runs) for F5
Function Evaluations, FEs
Digits
jDE100→ 10
DISHchain1e+12→ 10
HyDE-DF→ 10
SOMAT3A→ 10
ESHADE-USM→ 10
rCIPDE→ 10
Co-Op→ 10
DISH→ 10
mL-SHADE→ 10
GADE→ 9.44
CMEAL→ 10
HTPC→ 10
UMDE-MS→ 10
DLABC→ 10


Mean function evaluations vs. correct digits (over 25 runs) for F6
Function Evaluations, FEs
Digits
jDE100→ 10
DISHchain1e+12→ 10
HyDE-DF→ 10
SOMAT3A→ 10
ESHADE-USM→ 10
rCIPDE→ 10
Co-Op→ 10
DISH→ 10
mL-SHADE→ 10
GADE→ 10
CMEAL→ 10
HTPC→ 10
UMDE-MS→ 10
DLABC→ 10
ESP-SOMA→ 10


Mean function evaluations vs. correct digits (over 25 runs) for F7
Function Evaluations, FEs
Digits
jDE100→ 10
DISHchain1e+12→ 10
HyDE-DF→ 10
SOMAT3A→ 10
ESHADE-USM→ 10
rCIPDE→ 10
Co-Op→ 9.56
DISH→ 10
DLABC→ 10


Mean function evaluations vs. correct digits (over 25 runs) for F8
Function Evaluations, FEs
Digits
jDE100→ 10
DISHchain1e+12→ 10
HyDE-DF→ 10


Mean function evaluations vs. correct digits (over 25 runs) for F9
1e9
Function Evaluations, FEs
Digits
jDE100→ 10


Mean function evaluations vs. correct digits (over 25 runs) for F10
Function Evaluations, FEs
Digits
jDE100 → 10
DISHchain1e+12 → 10
HyDE-DF → 10
SOMAT3A → 10
ESHADE-USM → 10
rCIPDE → 10
Co-Op → 10
DISH → 10
mL-SHADE → 10
GADE → 10
CMEAL → 10
HTPC → 10
UMDE-MS → 10
DLABC → 10
ESP-SOMA → 10