COMP2024 Coursework

Group 11

1. Ng Jiun Loong(Group Leader)	20211830
2. Cheong Beng Chuan	20218995
3. Yong Zi Li	20106684
4. Ong Kwang Xi	20210392

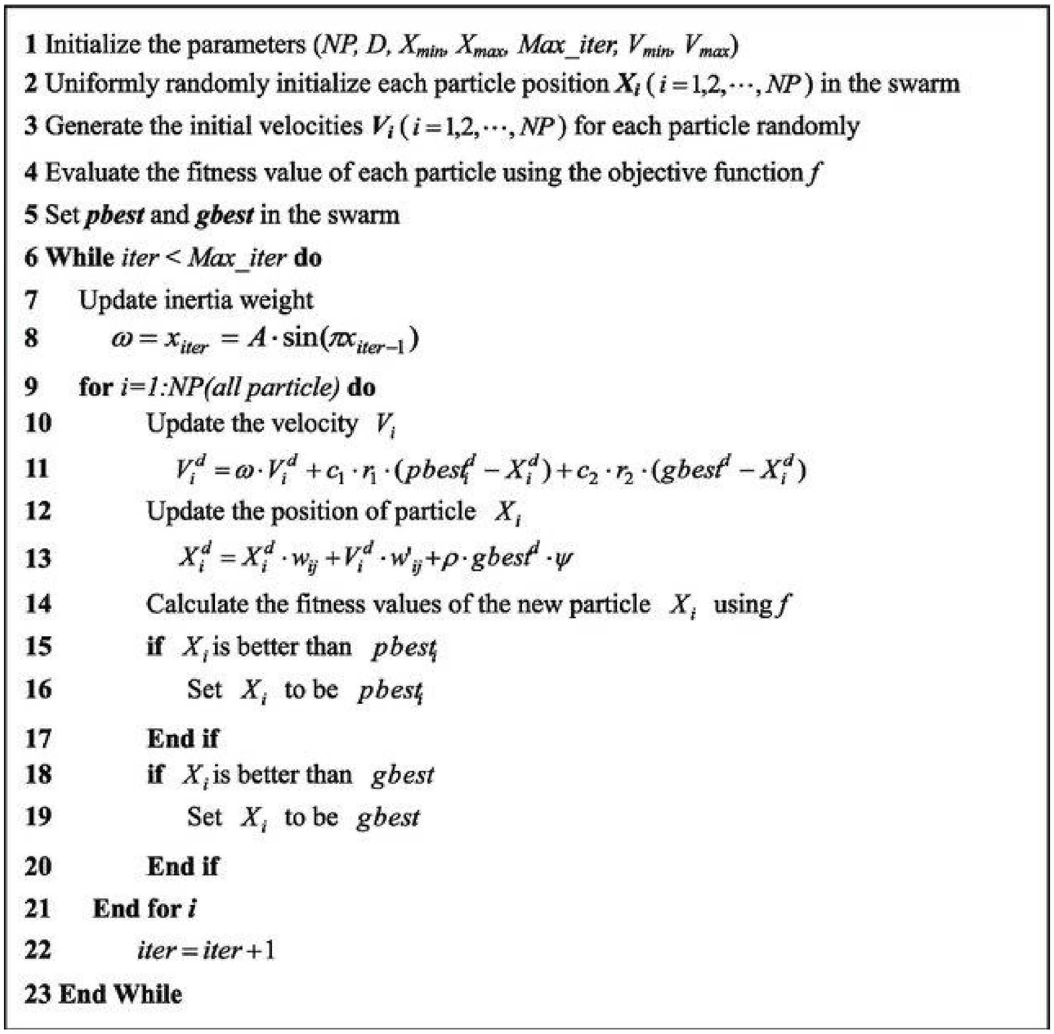
Optimizers

1. Particle Swarm Optimizer (PSO)

a. Introduction

Particle swarm optimization also known as PSO is a stochastic population-based optimization technique inspired by how a flock of birds moved. PSO is one of the most well regarded in the literature of single-objective optimization and it has also been used in many different industries and domains.

b. Algorithms



```
1 Initialize the parameters ($NP, D, X_{min}, X_{max}, Max\_iter, V_{min}, V_{max}$)
2 Uniformly randomly initialize each particle position $X_i$ ($i = 1,2,\cdots,NP$) in the swarm
3 Generate the initial velocities $V_i$ ($i = 1,2,\cdots,NP$) for each particle randomly
4 Evaluate the fitness value of each particle using the objective function $f$
5 Set pbest and gbest in the swarm
6 While $iter < Max\_iter$ do
7     Update inertia weight
8         $\omega = x_{iter} = A \cdot \sin(\pi x_{iter-1})$
9     for $i=1:NP$(all particle) do
10            Update the velocity $V_i$
11                $V_i^d = \omega \cdot V_i^d + c_1 \cdot r_1 \cdot (pbest_i^d - X_i^d) + c_2 \cdot r_2 \cdot (gbest^d - X_i^d)$
12            Update the position of particle $X_i$
13                $X_i^d = X_i^d \cdot w_{ij} + V_i^d \cdot w'_{ij} + \rho \cdot gbest^d \cdot \psi$
14            Calculate the fitness values of the new particle $X_i$ using $f$
15            if $X_i$ is better than $pbest_i$
16                Set $X_i$ to be $pbest_i$
17            End if
18            if $X_i$ is better than $gbest$
19                Set $X_i$ to be $gbest$
20            End if
21     End for $i$
22            $iter = iter + 1$
23 End While
```

Figure 1. PSO Pseudocode

The algorithm starts by initializing parameters. Then, it enters the While loop where the loop condition is if current iteration(iter) is less than the given maximum iteration(Max_iter). The algorithm will update the inertia weight(ω) after every iteration. ω is used to tune between exploitation and exploration in PSO. Next, it enters the For loop where every particle will update its Velocity(V_i^d, i = index of particle, d = iteration) which determines the particle's direction and its intensity of movement. Furthermore, it will update Position of the Particle (X_i^d). Then, the algorithm will use function F to calculate the fitness of each particle and see if the algorithm needs to update pbest and gbest. Finally, it will increment the iteration count.

c. Results(Runtime distributions (ECDFs) per function)

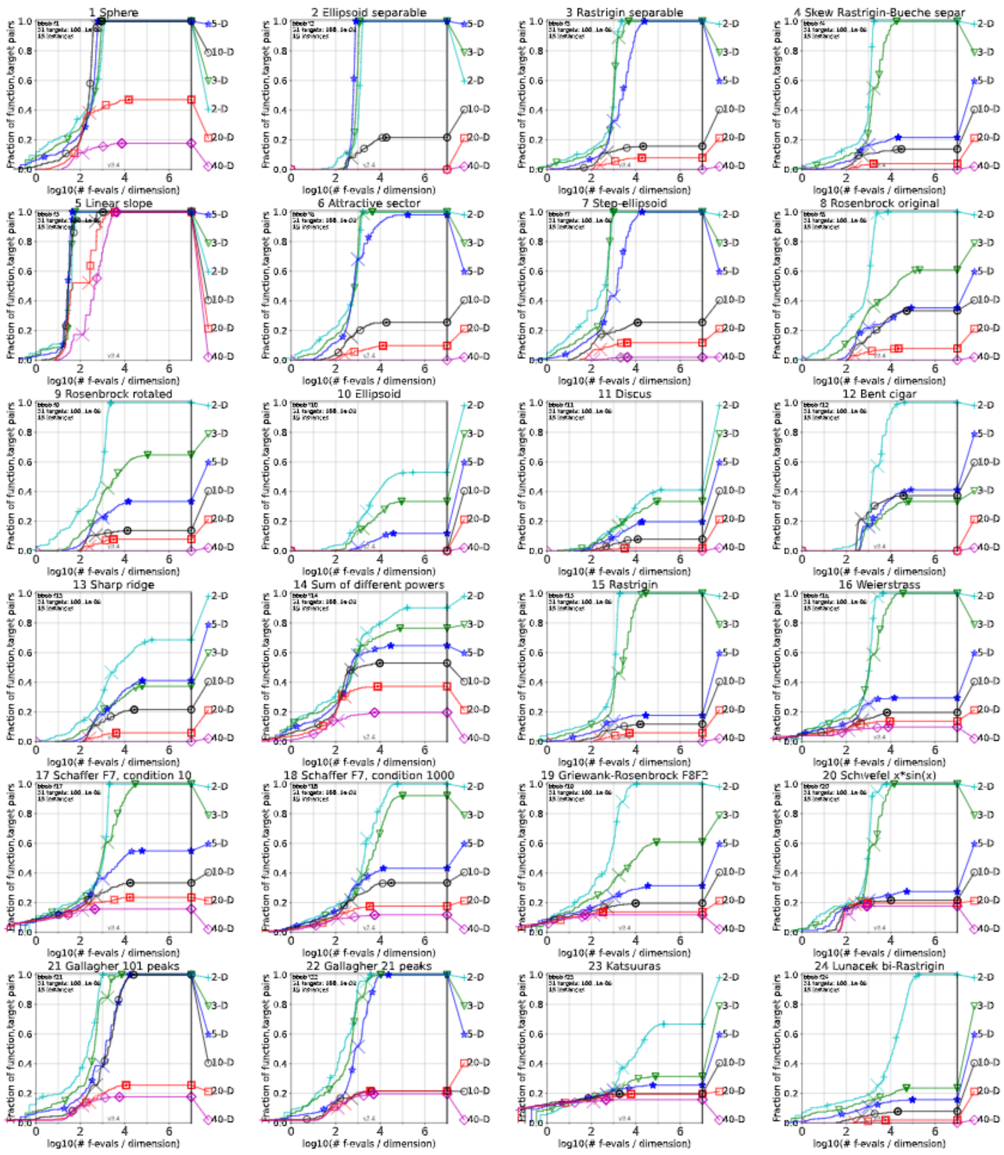


Figure 2. PSO Results

From Figure 2, we can see that PSO performed really well in lower dimensions such as 2,3,5 but its performance starts to plummet when the dimensions are above 10. Furthermore, it performed quite good in Sphere function, Linear Slope function and Gallagher 101 peaks function. Other than that, it performed okay in other functions except Ellipsoid function, Discus function, Sharp ridge function and Katsuuras function which the PSO struggles to get good results.

Parameters

popsize = 40; c1 = 1.4944; c2 = 1.4944; w = 0.792;

d. Reason of using it

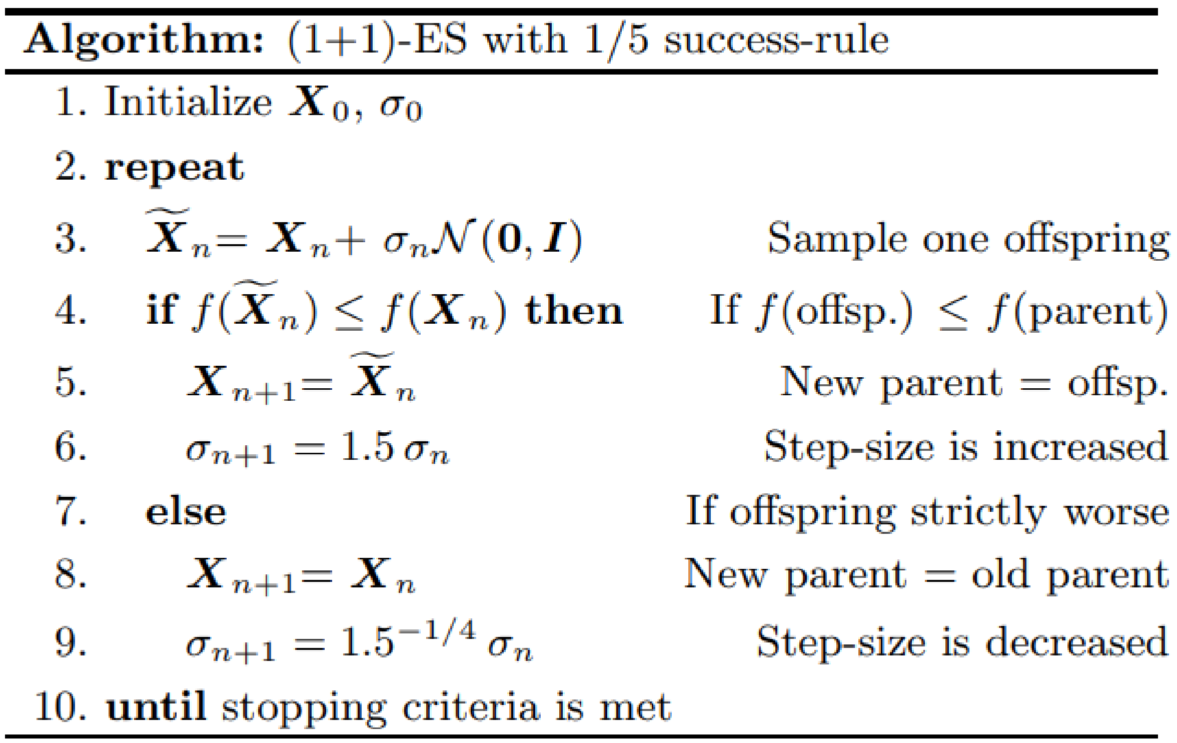
- Do not need sophisticated parameters tuning.
- The algorithm is straightforward and easy to implement.
- Has a high chance of finding the Global Optimum.
- Able to converge fast.
- Has short computation time.

2. Evolution Strategy with one-fifth success rule

a. Introduction

Evolution Strategy with one-fifth success rule is one of the first as well as simplest adaptive search algorithms. The one-fifth success rule was originally designed to automatically control the step size for evolution strategies. Basically, step size should increase when the probability of creating an offspring with a better fitness is more than or equal to 1/5, and decrease otherwise. The idea is that if there are too many offspring that are better than their parents, it might mean that the search is too local and therefore step-size should be increased, and vice versa.

b. Algorithms



Algorithm: (1+1)-ES with 1/5 success-rule

```
1. Initialize $\boldsymbol{X}_0$, $\sigma_0$
2. repeat
3.     $\widetilde{\boldsymbol{X}}_n = \boldsymbol{X}_n + \sigma_n \mathcal{N}(\mathbf{0}, \boldsymbol{I})$          Sample one offspring
4.     if $f(\widetilde{\boldsymbol{X}}_n) \leq f(\boldsymbol{X}_n)$ then          If $f$(offsp.) $\leq f$(parent)
5.         $\boldsymbol{X}_{n+1} = \widetilde{\boldsymbol{X}}_n$          New parent = offsp.
6.         $\sigma_{n+1} = 1.5\, \sigma_n$          Step-size is increased
7.     else          If offspring strictly worse
8.         $\boldsymbol{X}_{n+1} = \boldsymbol{X}_n$          New parent = old parent
9.         $\sigma_{n+1} = 1.5^{-1/4}\, \sigma_n$          Step-size is decreased
10. until stopping criteria is met
```

Figure 3. ES Pseudocode

The algorithm starts by initializing X_0, the parent, as well as step-size. Then, for each iteration, generate an offspring by adding the parent to the product of step-size and a random number derived from a multivariate normal distribution. If the fitness value of the offspring is better than or equal to the parent, the offspring will become the new parent and the step-size is increased by a factor of 1.5. If not, the parent will become the new parent, and the step-size will decrease by a factor of $1.5^{-\frac{1}{4}}$.

c. Results(Runtime distributions (ECDFs) per function)

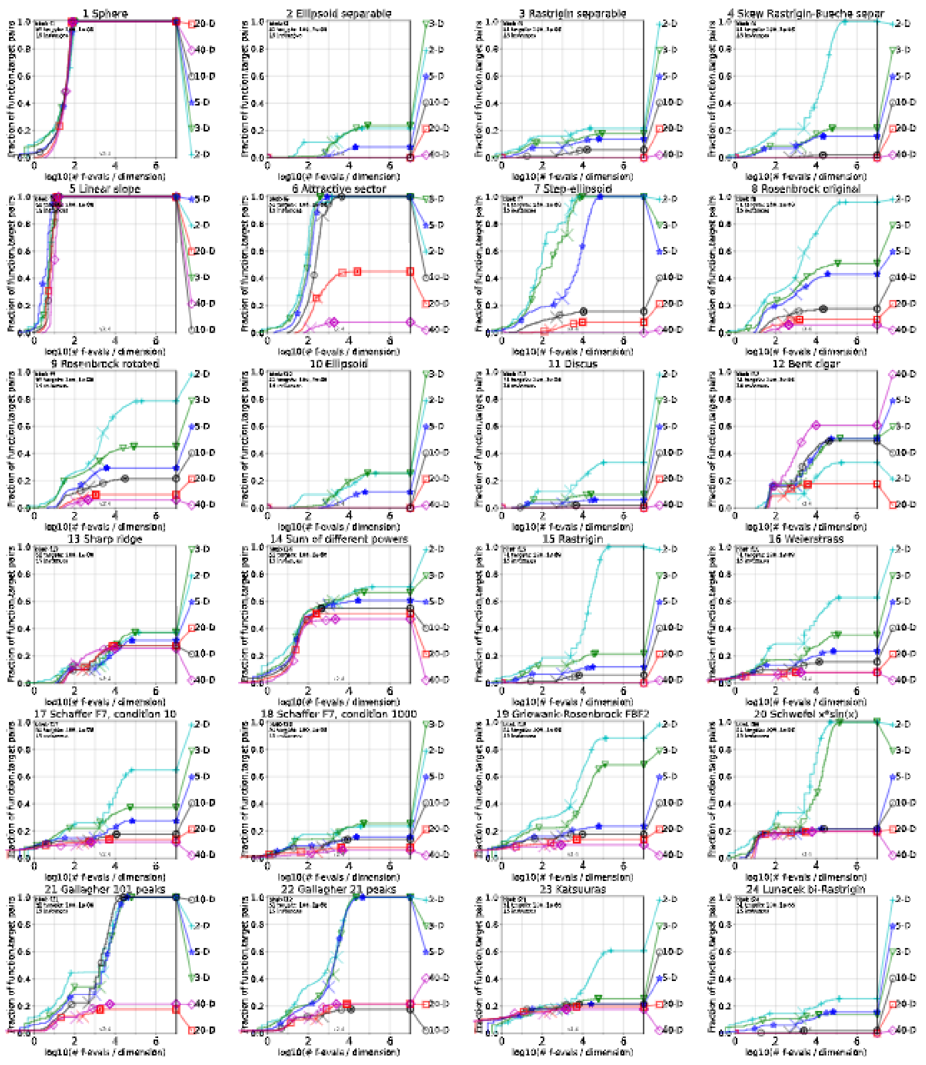


Figure 4. ES Results

From Figure 4, we can see that ES Strategy performance is on par with CMA-ES in a couple functions, like Sphere function, Linear slope function, and Attractive sector function. It fares well against CMA-ES in Step-ellipsoid function, Schwefel function and Gallagher 101 peaks function.

d. Reason of using it

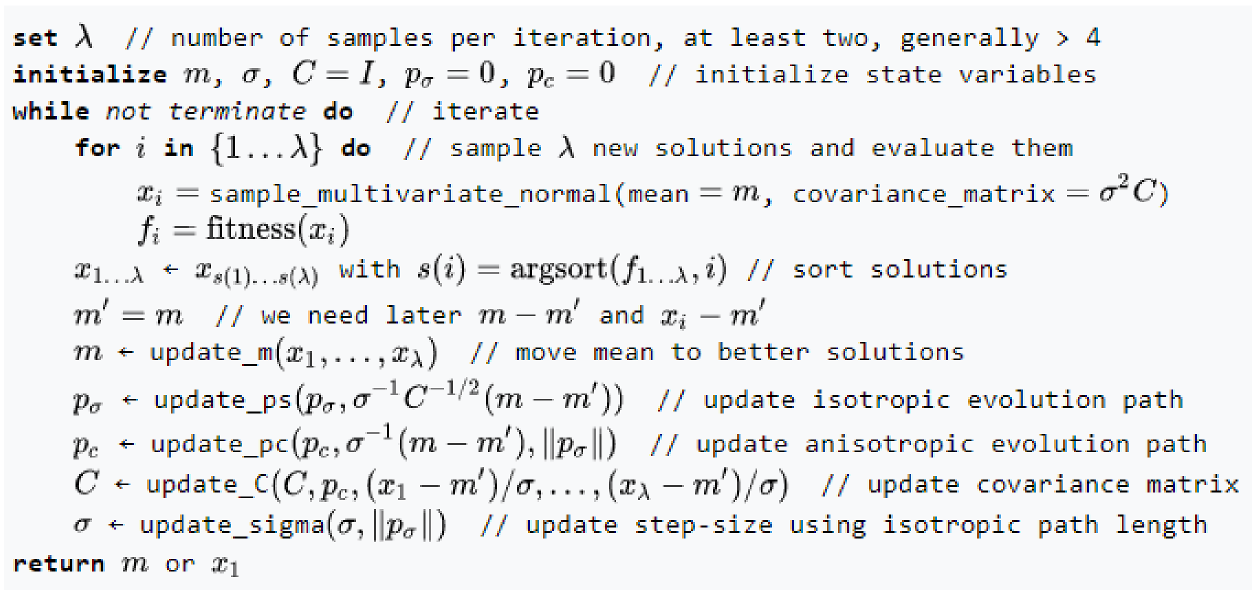
- Capable of finding optimal or close to optimal parameter settings
- Simple to implement
- Wide array of uses as it does not require problem-specific knowledge

3. Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

a. Introduction

CMA-ES is a variety of evolutionary algorithm just like the previous mentioned Evolution Strategy. The CMA-ES is arguably the cream of the crop in evolutionary computation and has been used as one of the standard tools for continuous optimisation in many different fields around the world. There are a few intriguing characteristics of CMA-ES. The first being CMA-ES has several invariance properties, Invariances are highly sought after the reason being they indicate the consistent behavior on classes of functions and therefore signify the generalization of empirical results. Secondly, CMA-ES does not need parameter tuning for its application. The strategy parameters are part of the algorithm design itself, the goal is to have a good performing algorithm at the very start without much parameters tweaking.

b. Algorithms



```
set $\lambda$  // number of samples per iteration, at least two, generally > 4
initialize $m$, $\sigma$, $C = I$, $p_\sigma = 0$, $p_c = 0$  // initialize state variables
while not terminate do  // iterate
    for $i$ in $\{1 \ldots \lambda\}$ do  // sample $\lambda$ new solutions and evaluate them
        $x_i$ = sample_multivariate_normal(mean $= m$, covariance_matrix $= \sigma^2 C$)
        $f_i = \text{fitness}(x_i)$
    $x_{1\ldots\lambda}$ ← $x_{s(1)\ldots s(\lambda)}$ with $s(i) = \text{argsort}(f_{1\ldots\lambda}, i)$  // sort solutions
    $m' = m$  // we need later $m - m'$ and $x_i - m'$
    $m$ ← update_m$(x_1, \ldots, x_\lambda)$  // move mean to better solutions
    $p_\sigma$ ← update_ps$(p_\sigma, \sigma^{-1} C^{-1/2}(m - m'))$  // update isotropic evolution path
    $p_c$ ← update_pc$(p_c, \sigma^{-1}(m - m'), \|p_\sigma\|)$  // update anisotropic evolution path
    $C$ ← update_C$(C, p_c, (x_1 - m')/\sigma, \ldots, (x_\lambda - m')/\sigma)$  // update covariance matrix
    $\sigma$ ← update_sigma$(\sigma, \|p_\sigma\|)$  // update step-size using isotropic path length
return $m$ or $x_1$
```

Figure 5. CMA-ES Pseudocode

The algorithm begins with initializing state variables. Then, it enters the While loop which within this loop, all it basically does is sampling of the new solutions, sorting according to their fitness and updating internal variables based on the previously sorted sample.

c. Results(Runtime distributions (ECDFs) per function

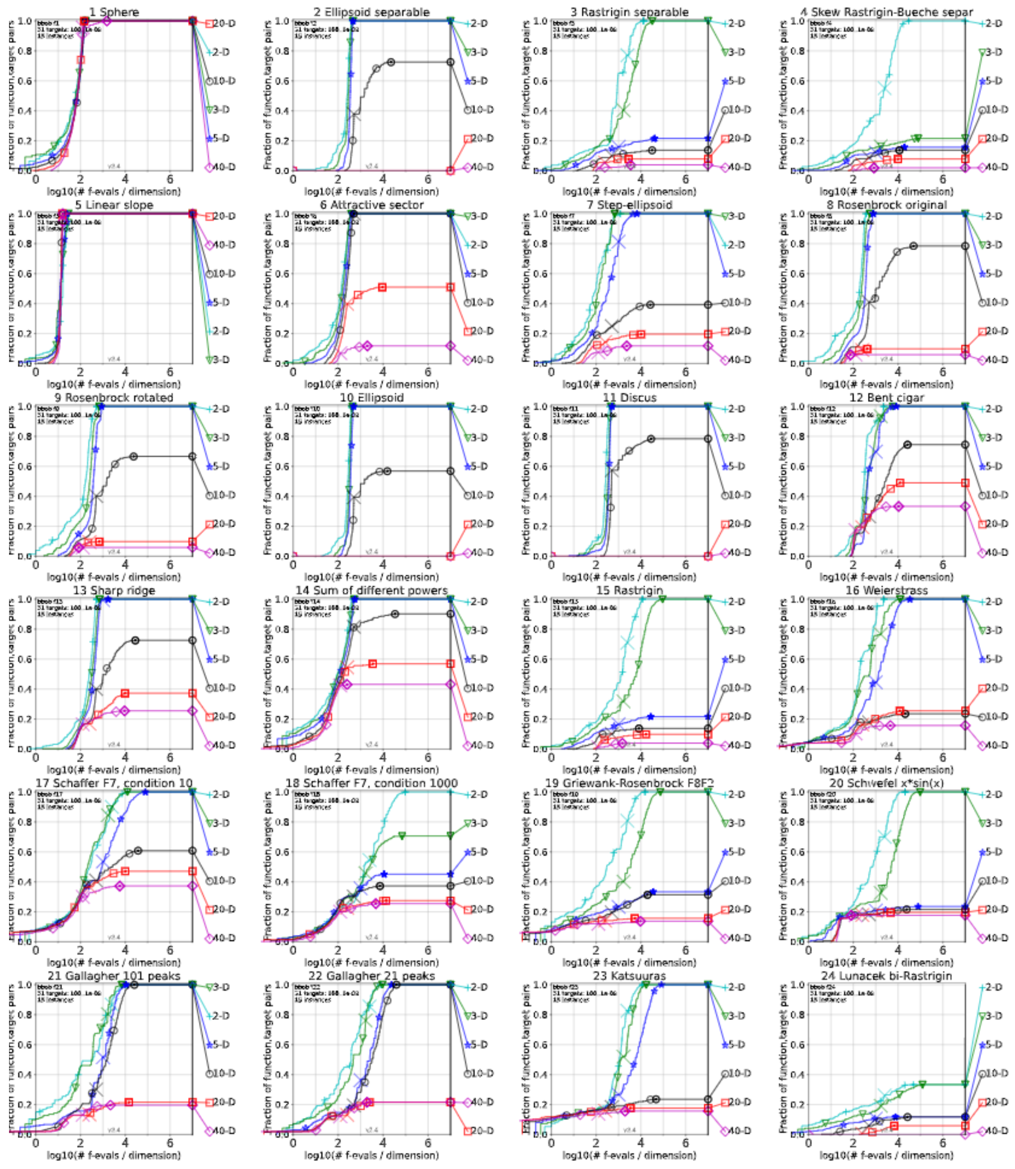


Figure 6. CMA-ES Results

From Figure 6, we can observe that CMA-ES is able to produce the best results out of the 3 optimizers we reviewed in this report. It performed really well in a lot of functions except Rastrigin separable function, Skew Rastigin-Bueche separable function, Rastrigin function, Schaffer F7 function, Schwefel function and Lunacek bi-Rastrigin which it produced mediocre results. Besides that, It also has a similar issue as PSO but not as severe as it, its performance is not ideal in higher dimensional functions.

d. Reason of using it

- Do not need sophisticated parameters tuning.

- Performance is really good in almost every functions
- Has invariance properties
- Workable on non-separable problems or badly conditioned problems

Comparison Of All Reviewed Optimizers

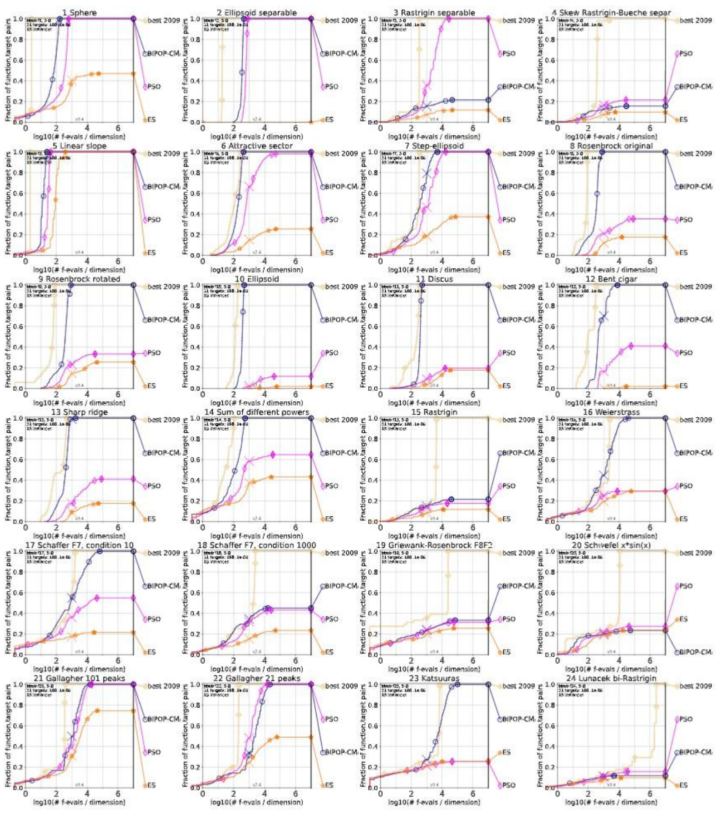


Figure 7. All Reviewed Optimizers Comparison(5 Dimensions)

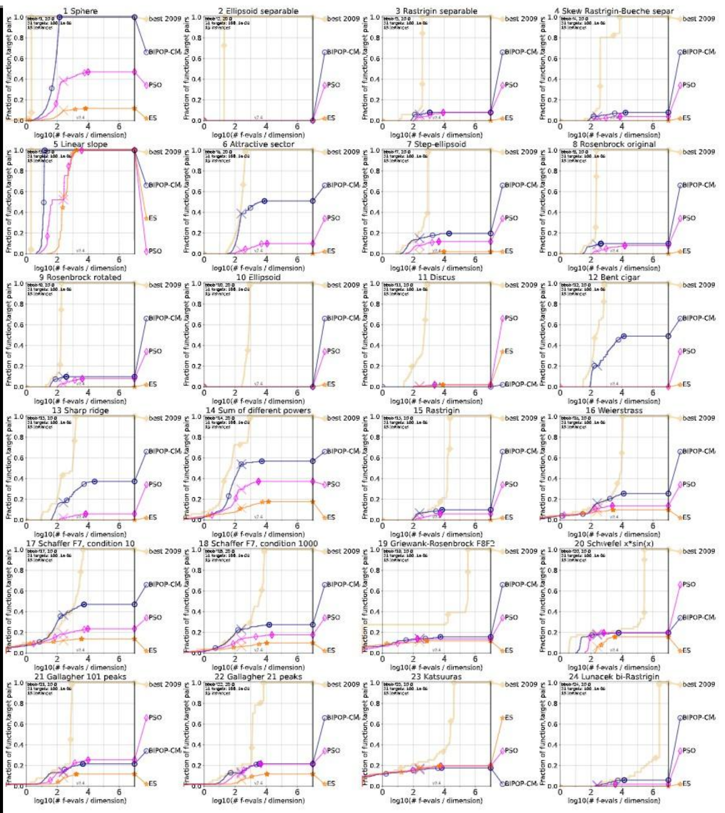


Figure 8. All Reviewed Optimizers Comparison(20 Dimensions)

From Figure 7 and 8, we are able to compare the performance between all reviewed optimizers (PSO, ES, CMA-ES*(BIPOP CMA-ES)*) in this report and also the 2009 state of the art algorithm(best 2009) is included as well. In Figure 7, we can see that CMA-ES performed better than PSO and ES for the most part but performed worse than best 2009. There is an interesting point to highlight in Figure 7 where PSO performed really well in the Rastrigin Separable function even better than CMA-ES but not as good as best 2009. In Figure 8, we can see that best 2009 has a great lead in almost every function in the 20D setting. However, in some scenarios, CMA-ES is still able to produce decent results such as in Sphere function, Attractive sector and Sum of Different powers. Otherwise, all reviewed optimizers are not able to keep up with best 2009 and perform poorly or below average results. From this analysis, we can conclude that out of our all three reviewed optimizers, CMA-ES performed the best when considering the overall performance, therefore it is our choice of optimizer for the competition.

References

1. Mohammed El-Abd & Mohamed S. Kamel (July 2009).
   Black-Box Optimization Benchmarking for Noiseless Function Testbed using Particle Swarm Optimization.
   https://sci2s.ugr.es/sites/default/files/files/TematicWebSites/EAMHCO/contributionsGECCO09/p2269-elabd.pdf

2. Ke Chen, Fengyu Zhou and Aling Liu (October 2017).
   Chaotic Dynamic Weight Particle Swarm Optimization for Numerical Function Optimization.
   https://www.researchgate.net/publication/320365527_Chaotic_Dynamic_Weight_Particle_Swarm_Optimization_for_Numerical_Function_Optimization

3. Anne Auger (July 2009).
   Benchmarking the (1+1) Evolution Strategy with One-Fifth Success Rule on the BBOB-2009 Function Testbed.
   https://sci2s.ugr.es/sites/default/files/files/TematicWebSites/EAMHCO/contributionsGECCO09/p2447-auger.pdf

4. Benjamin Doerr & Carola Doerr(April 2015).
   Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings.
   https://arxiv.org/pdf/1504.03212.pdf

5. Nikolaus Hansen (2016).
   The CMA Evolution Strategy.
   http://cma.gforge.inria.fr/

6. Petr Pošík & Václav Klemš (2012).
   Benchmarking the Differential Evolution with Adaptive Encoding on Noiseless Functions.
   http://www.cmap.polytechnique.fr/~nikolaus.hansen/proceedings/2012/GECCO/companion/p189.pdf