Olive Ridley Survival Optimizer

Improving performance using Alpha and Beta Parameters

Instructor Name: Dr. Ankur Gupta

Authors: Vishwesh Kumar Gupta (2023UCA1922), Harry Bharti (2023UCA1948), Ayush Bhardwaj (2023UCA1951)

Period: 4^{th} Semester: 2024–2025

Class: CSAI S-2, 2023–2027 Batch

Abstract

The Olive Ridley Survival Optimizer (ORSO) is a nature-inspired metaheuristic algorithm modeled after the nesting and emergence behavior of Olive Ridley turtles. This document presents the Python implementation of ORSO, including its core logic, biological parameters, optimization performance, and visualizations of convergence across multiple runs. Enhancements using adaptive learning strategies with `alpha` and `beta` are also included to improve exploration-exploitation balance.

Contents

1 Theory of the Olive Ridley Survival Optimizer

Basics: In this paper, a novel modification to the meta-heuristic optimization algorithm, Olive Ridley Survival (ORS), is proposed which is inspired by the survival challenges faced by the hatchlings of Olive Ridley sea turtles. A major fact about the survival of Olive Ridley turtles reveals that out of 1,000 Olive Ridley hatchlings which emerge from nest, only one survives at sea due to various environmental and other factors. This fact acts as the backbone for developing the proposed algorithm. The algorithm has two major phases: hatchling survival through environmental factors and impact of movement trajectory on its survival. The phases are mathematically modeled and implemented along with a suitable input representation and fitness function. The algorithm is theoretically analyzed. To modify the algorithm. the effect of the temperature has been taken to be exponential to speed. The time of the day acts as a sigmoid function, and emergence is momentum. The turtles follow the global best solution. Only the random exploitation of the original ORSO has been kept the same; the rest of the linears have been changed.

Olive Ridley LIfe Cycle Behaviour: The Olive Ridley sea turtle, scientifically known as Lepidochelys olivacea, is both the most numerous and diminutive of the seven kinds of turtles found on Earth. Because of the direct effect they have on other marine life,sea turtles are thought to be essential to the health of the ocean. Thus, turtles are generally thought of as an integral component of marine ecosystems. Olive ridleys are known for their "arribadas" (Spanish for "arrivals"), which are mass nesting events that only occur in a few unique topography locations of Odisha's Rusikulya, Gahiramatha, and chandrabhaga in India, Costa Rica, and Mexico. Olive ridleys have a worldwide distribution that encompasses the entire circumtropical region. In the course of this spectacular event, thousands of female turtles congregate at the same time on particular nesting beaches in order to lay their eggs. This coordinated nesting behavior produces a captivating show along the shoreline, usually at night. The turtles painstakingly excavate their nests in the fine sand, laying a profusion of eggs and then sanding them down. The migratory behavior exhibited by numerous Olive Ridley turtle populations is an effective biological strategy for efficiently locating prey that is unevenly distributed. Olive ridleys are the most common sea turtles, which may be explained by their adaptability to the sudden changes in dynamic habitats. Nevertheless, sea turtle numbers worldwide have been steadily decreasing for several decades. As a result of this significant decrease, all turtle species have been categorized as vulnerable, endangered, or critically endangered within the marine ecosystem. Olive ridleys encounter several hazards while they are near the sea. Thus, the loss of natural nesting habitats for olive ridleys is mostly caused by casuarina plantations, beach erosion, the development of tourist complexes, artificial lighting, and egg and hatchling predation. Olive ridleys are vulnerable to a number of hazards in oceanic environments, including pollution, harmful algal blooms, directed fishing net capture, bycatch, and marine trash, all of which have the potential to be lethal.

Survival Phenomena: The Olive Ridley sea turtle has evolved a holistic strategy to survival by utilizing a combination of reproductive, behavioral, and physiological tactics. This demonstrates its impressive flexibility to the obstacles posed by both terrestrial and marine habitats. Mass Nesting (Arribada): One of the unique survival tactics employed by Olive Ridleys is the phenomena known as "Arribada," or mass nesting. The coordi-

nated nesting habit of hundreds of female turtles entails their simultaneous convergence on particular beaches to deposit their eggs. By overwhelming predators, this tactic raises the likelihood that some eggs will hatch. Egg Placement and Camouflage: The female Olive Ridley meticulously chooses and excavates nests in the sandy beaches, delicately places their eggs and cover these with sand to disguise them. The choice of strategically hidden nesting locations aids in shielding the eggs from potential predators and environmental hazards. Temperature-Dependent Sex Determination: The Olive Ridley sea turtle exhibits temperature-dependent sex determination. The sex of the hatchlings is influenced by the ambient temperature during incubation; warmer temperatures produce more females. The species is more resilient to changes in its environment because of its adaptation. Habitat Selection: Olive Ridley turtles demonstrate a discerning choice of environments for the purposes of foraging, mating, and nesting. The selection of these options is determined by various circumstances, including water temperature, prey abundance, and appropriate nesting locations, which collectively contribute to the species' overall survival and reproductive achievements.

Motivation: Despite of several natural survival tactics adapted by this species, during birth phase, Olive Ridley turtles known as hatchlings, strive to survive due to various environmental and other indirect factors. One alarming observation is that out of one thousand Olive Ridley hatchlings which emerge from nest, only one survive at sea Burger and Gochfeld [2014]. Once, hatchlings emerge from their nests, they start crawling towards sea as shown in Fig. 3. During their movement towards sea, various environmental factors, e.g., sand temperature, emergence order from nests, and time of the day of the crawling affect their speed of movement Burger and Gochfeld [2014]. They suffer thermal stress during high temperature period of the day which causes death of some hatchlings. Hatchlings emergence order from nests is also a major factor of their momentum towards sea which affects their survival. The hatchlings emerging early crawl faster than hatchlings emerging late. In case of slower speed, the hatchlings have high chance of exposure to land predators and avian. Another indirect factor that affects locomotion of hatchlings is their movement trajectory. During their movement on sea shore, they face debris and as a result, they change their direction of movement. If they deviate from straight line movement, time to reach to sea increases and they are supposed to expose towards predators. In summary, hatchlings' survival depends on how fast they can reach to sea. But, their survival is affected by their crawling speed and in turn, affect their fitness. This motivates the development of the proposed meta-heuristic, and its subsequent modification. The modification is mainly based on the introduction of the Alpha and Beta parameters, which significantly increase the performance of the optimizer on various metrics, as shown subsequently.

2 ORSO: Olive Ridley Survival Optimizer

Based on the base paper

2.1 Initial Version

```
import numpy as np
class OliveRidleySurvivalOptimizer:
    def __init__(self, objective_function, num_turtles=30,
       dimensions=2, lower_bound=-100, upper_bound=100,
                 max_iterations=1000, tolerable_temp=28, max_temp
                    =35, time_range=(6, 18)):
        self.obj_func = objective_function
        self.num_turtles = num_turtles
        self.dimensions = dimensions
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.max_iterations = max_iterations

        # Initialize turtle positions and velocities
        self.velocities = np.random.uniform(lower_bound,
           upper_bound, (num_turtles, dimensions)).astype(np.
           float64)
        self.turtles = np.random.uniform(-1, 1, (num_turtles,
           dimensions)).astype(np.float64)

        self.best_turtle = None
        self.best_fitness = float("inf")

        # Biological parameters
        self.sand_temperature = np.random.uniform(25, 38,
           num_turtles).astype(np.float64)
        self.time_of_day = np.random.uniform(0, 24, num_turtles).
           astype(np.float64)
        self.emergence_types = np.random.choice([-1, 0, 1],
           num_turtles).astype(np.float64)

        self.tolerable_temp = tolerable_temp
        self.max_temp = max_temp
        self.time_range = time_range

        self.convergence = []

    def temperature_impact(self, temp):
        """ Linear speed adjustment based on temperature """
        if temp < self.tolerable_temp:
            return 2  # Linear increase
        elif self.tolerable_temp <= temp <= self.max_temp:
            return 1  # Linear decrease
        else:
```

```
            return -np.inf  # Turtle dies

    def time_of_day_impact(self, time):
        return 2 if self.time_range[0] <= time <= self.time_range
            [1] else 1

    def emergence_impact(self, emergence_type):
        return emergence_type  # Early (+1), Middle (0), Late
            (-1)
```

```
    def optimize(self):
        for iteration in range(self.max_iterations):
            fitness_values = np.apply_along_axis(self.obj_func,
                1, self.velocities)
            max_index = np.argmax(fitness_values)
            worst_fitness = fitness_values[max_index]
            worst_turtle = self.velocities[max_index].copy()
            min_index = np.argmin(fitness_values)
            if fitness_values[min_index] < self.best_fitness:
                self.best_fitness = fitness_values[min_index]
                self.best_turtle = self.velocities[min_index].
                    copy()

            self.convergence.append(self.best_fitness)

            # Exploration
            random_factor = np.random.uniform(0, 1, (self.
                num_turtles, self.dimensions))
            movement_direction = np.sign(np.random.uniform(-1, 1,
                (self.num_turtles, self.dimensions)))
            exploration_step = movement_direction * random_factor
            new_velocities = np.zeros((self.num_turtles, self.
                dimensions))
            new_velocities += exploration_step


            for i in range(self.num_turtles):
                temp_factor = np.float64(self.temperature_impact(
                    self.sand_temperature[i]))
                time_factor = np.float64(self.time_of_day_impact(
                    self.time_of_day[i]))
                emergence_factor = np.float64(self.
                    emergence_impact(self.emergence_types[i]) * 2
                    + 1*self.velocities[i])

                if temp_factor == -np.inf:
                    fitness_values[i] = np.inf
                    continue

                velocity_change_temp = self.velocities[i] *
                    temp_factor
                velocity_change_time = self.velocities[i] *
```

```
                    time_factor
                velocity_change_emergence = emergence_factor

                new_velocities[i] += velocity_change_temp +
                    velocity_change_time +
                    velocity_change_emergence
                if ((fitness_values[i]-self.best_fitness)/(
                    worst_fitness-self.best_fitness)<=0.3):
                  self.velocities[i] += self.best_turtle -
                      new_velocities[i]
                else:
                  self.velocities[i] += self.best_turtle +
                      new_velocities[i]
                self.velocities = np.clip(self.velocities, self.
                    lower_bound, self.upper_bound)

            #if (iteration + 1) % 100 == 0:
                #print(f"Iteration {iteration + 1}, Best Fitness:
                    {self.best_fitness}")

        return self.best_turtle, self.best_fitness, self.
            convergence
```

2.2 Example Usage of Initial Version

```
# Example Usage
def sphere_function(x):
  return np.sum(x**2)  # Sphere function (min at x = 0)
# Run ORSO
for i in range(5):
  orso = OliveRidleySurvivalOptimizer(objective_function=
      sphere_function, dimensions=5, max_iterations=100)
  best_solution, best_fitness,convergence = orso.optimize()
  print("\nBest Solution:", best_solution)
  print("Best Fitness:", best_fitness)
  print()
```

2.3 Output (Sample) of Initial Version

```
Best Solution: [ -2.07589314 -25.28824984  47.89912529  -4.59939069 -16.30255926]
Best Fitness: 3225.0589483618196

Best Solution: [13.59033866  6.00341023 24.43874123  2.70658809 13.13077367]
Best Fitness: 997.7331486418366

Best Solution: [-22.6768817    4.84039593  56.46068533  54.3013319  -17.84546922]
Best Fitness: 6992.574802997446

Best Solution: [ 30.08082683 -20.77681677 -18.17319821  10.28981968  48.01362084]
```

```
Best Fitness: 4077.9855658628476

Best Solution: [-0.45691295  0.83597343 -2.07048312 -0.53574248 -0.11593992]
Best Fitness: 5.494983450397474
```

2.4 Best Variation (Bio hybrid ORSO): (temp effect is exponential to speed, time of day acts as sigmoid, emergence is momentum, like turtles follow global best solution, only random exploitation kept same, rest linears are changed to above)

```
class ModifiedOliveRidleySurvivalOptimizer:
    def __init__(self, objective_function, num_agents=30,
       dimensions=5, lower_bound=-100, upper_bound=100,
                 max_iterations=500, tolerable_temp=28, max_temp
                    =35, time_range=(6, 18),
                 beta=0.6, learning_rate=0.05):
        self.obj_func = objective_function
        self.num_agents = num_agents
        self.dimensions = dimensions
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.max_iterations = max_iterations

        self.beta = beta
        self.learning_rate = learning_rate

        self.positions = np.random.uniform(lower_bound,
           upper_bound, (num_agents, dimensions))
        self.velocities = np.random.uniform(-1, 1, (num_agents,
           dimensions))

        self.best_score = float("inf")
        self.best_position = None

        self.temperatures = np.random.uniform(25, 38, num_agents)
        self.times = np.random.uniform(0, 24, num_agents)
        self.emergence = np.random.choice([-1, 0, 1], num_agents)

        self.tolerable_temp = tolerable_temp
        self.max_temp = max_temp
        self.time_range = time_range

    def temperature_impact(self, temp):
        if temp < self.tolerable_temp:
            return 1 + np.exp(-0.1 * (self.tolerable_temp - temp)
               )
        elif self.tolerable_temp <= temp <= self.max_temp:
            return np.exp(-0.1 * (temp - self.tolerable_temp))
        else:
            return -np.inf

    def time_of_day_impact(self, t):
        midpoint = (self.time_range[0] + self.time_range[1]) / 2
        steepness = 0.5
```

```
        return 1 / (1 + np.exp(-steepness * (t - midpoint))) +
           0.5

    def emergence_impact(self, e):
        return 1.0 + 0.2 * e
```

```
    def optimize(self):
        convergence = []

        for iteration in range(self.max_iterations):
            fitness = np.apply_along_axis(self.obj_func, 1, self.
               positions)
            min_idx = np.argmin(fitness)

            if fitness[min_idx] < self.best_score:
                self.best_score = fitness[min_idx]
                self.best_position = self.positions[min_idx].copy
                   ()

            convergence.append(self.best_score)

            exploration_decay = 1.0 - iteration / self.
               max_iterations
            random_noise = exploration_decay * np.random.uniform
               (-1, 1, (self.num_agents, self.dimensions))
            self.positions += random_noise

            for i in range(self.num_agents):
                temp = self.temperature_impact(self.temperatures[
                   i])
                time = self.time_of_day_impact(self.times[i])
                emerge = self.emergence_impact(self.emergence[i])

                if temp == -np.inf:
                    fitness[i] = np.inf
                    continue

                bio_influence = temp * time * emerge
                cognitive_component = np.random.rand(self.
                   dimensions) * (self.best_position - self.
                   positions[i])
                velocity_update = self.beta * self.velocities[i]
                   + (1 - self.beta) * cognitive_component
                velocity_update *= bio_influence

                self.velocities[i] = velocity_update
                self.positions[i] += velocity_update

            self.positions = np.clip(self.positions, self.
               lower_bound, self.upper_bound)
```

```
            imp_rate = abs(self.best_score - np.min(fitness)) / (
                abs(self.best_score) + 1e-8)
            self.beta = max(0.3, min(0.9, self.beta * (1 - self.
                learning_rate * imp_rate)))

            if (iteration + 1) % 100 == 0:
                print(f"Iteration {iteration+1} | Best Fitness: {
                    self.best_score:.4f} | Beta: {self.beta:.3f}")

        return self.best_position, self.best_score, convergence
```

2.5 Example Usage of Best Version

```
def sphere(x):
    return np.sum(x**2)

convs = []
for run in range(5):
    print(f"\n--- Run {run + 1} ---")
    orso = ModifiedOliveRidleySurvivalOptimizer(
        objective_function=sphere, dimensions=5, max_iterations
        =500)
    best_sol, best_fit, conv = orso.optimize()
    print(f"Best Fitness: {best_fit}")
    convs.append(conv)
```

2.6 Output (Sample)

```
--- Run 1 ---
Iteration 100 | Best Fitness: 0.0204 | β: 0.300
Iteration 200 | Best Fitness: 0.0091 | β: 0.300
Iteration 300 | Best Fitness: 0.0088 | β: 0.300
Iteration 400 | Best Fitness: 0.0039 | β: 0.300
Iteration 500 | Best Fitness: 0.0000 | β: 0.300
Best Fitness: 3.761764702051026e-06

--- Run 2 ---
Iteration 100 | Best Fitness: 0.0368 | β: 0.300
Iteration 200 | Best Fitness: 0.0024 | β: 0.300
Iteration 300 | Best Fitness: 0.0024 | β: 0.300
Iteration 400 | Best Fitness: 0.0024 | β: 0.300
Iteration 500 | Best Fitness: 0.0000 | β: 0.300
Best Fitness: 1.231065792747576e-05

--- Run 3 ---
Iteration 100 | Best Fitness: 0.0607 | β: 0.300
Iteration 200 | Best Fitness: 0.0311 | β: 0.300
Iteration 300 | Best Fitness: 0.0104 | β: 0.300
```

Iteration 400 | Best Fitness: 0.0020 | β : 0.300
Iteration 500 | Best Fitness: 0.0000 | β : 0.300
Best Fitness: 1.1125437938587747e-05

--- Run 4 ---
Iteration 100 | Best Fitness: 0.0227 | β : 0.300
Iteration 200 | Best Fitness: 0.0079 | β : 0.300
Iteration 300 | Best Fitness: 0.0037 | β : 0.300
Iteration 400 | Best Fitness: 0.0023 | β : 0.300
Iteration 500 | Best Fitness: 0.0000 | β : 0.300
Best Fitness: 3.496032015471339e-05

--- Run 5 ---
Iteration 100 | Best Fitness: 0.0298 | β : 0.300
Iteration 200 | Best Fitness: 0.0138 | β : 0.300
Iteration 300 | Best Fitness: 0.0059 | β : 0.300
Iteration 400 | Best Fitness: 0.0050 | β : 0.300
Iteration 500 | Best Fitness: 0.0000 | β : 0.300
Best Fitness: 1.237161737756893e-05

3 Variants to the ORSO

Variants act similarly to the original, but the optimized one includes all their improvements and becomes a better model. The analysis of the original vs. modified ORSO is shown in the static analysis section

3.1 Variation 1: Exponent based Variable Temperature

```
import numpy as np

class OliveRidleySurvivalOptimizer:
    def __init__(self, objective_function, num_turtles=30,
       dimensions=2, lower_bound=-5, upper_bound=5,
                 max_iterations=100, tolerable_temp=28, max_temp
                    =35, time_range=(6, 18)):
         self.obj_func = objective_function
        self.num_turtles = num_turtles
        self.dimensions = dimensions
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.max_iterations = max_iterations
        # Initialize turtle positions and velocities
        self.turtles = np.random.uniform(lower_bound, upper_bound
           , (num_turtles, dimensions))
        self.turtles=np.array(self.turtles,dtype=np.float64)
        self.velocities = np.random.uniform(-1, 1, (num_turtles,
           dimensions))  # Initial velocity
        self.velocities=np.array(self.velocities,dtype=np.float64
           )
        self.best_turtle = None
        self.best_fitness = float("inf")
        # Biological parameters
        self.sand_temperature = np.random.uniform(25, 38,
           num_turtles)  #Random temp between 25 C  and 38 C
        self.sand_temperature=np.array(self.sand_temperature,
           dtype=np.float64)
        self.time_of_day = np.random.uniform(0, 24, num_turtles)
            # Time in hours (0 to 24)
        self.time_of_day=np.array(self.time_of_day,dtype=np.
           float64)
        # Emergence types (early: +1, middle: 0, late: -1)
        self.emergence_types = np.random.choice([-1, 0, 1],
           num_turtles)
        self.emergence_types=np.array(self.emergence_types,dtype=
           np.float64)
        # Constants
        self.tolerable_temp = tolerable_temp
        self.max_temp = max_temp
        self.time_range = time_range  # (start_time, end_time)
    def temperature_impact(self, temp):
```

```
        """ Adjust speed based on temperature conditions """
        if temp < self.tolerable_temp:
            return 1 + np.exp(-0.1 * (self.tolerable_temp - temp)
               )  # Speed increases
        elif self.tolerable_temp <= temp <= self.max_temp:
            return np.exp(-0.1 * (temp - self.tolerable_temp))  #
                Speed decreases
        else:
            return -np.inf  # Turtle dies
    def time_of_day_impact(self, time):
        """ Speed impact based on time of day """
        return 1.5 if self.time_range[0] <= time <= self.
           time_range[1] else 0.5
    def emergence_impact(self, emergence_type):
        """ Speed impact based on emergence order """
        return emergence_type  # Early (+1), Middle (0), Late
           (-1)
```

```
    def optimize(self):
        for iteration in range(self.max_iterations):
             fitness_values = np.apply_along_axis(self.obj_func,
                1, self.turtles)
            # Update best solution found
            min_index = np.argmin(fitness_values)
            if fitness_values[min_index] < self.best_fitness:
                self.best_fitness = fitness_values[min_index]
                self.best_turtle = self.turtles[min_index].copy()
            # --- Exploration: Change in movement trajectory --
            random_factor = np.random.uniform(0, 1, (self.
               num_turtles, self.dimensions))
            movement_direction = np.sign(np.random.uniform(-1, 1,
                (self.num_turtles, self.dimensions)))
            exploration_step = (1 - iteration / self.
               max_iterations) * movement_direction *
               random_factor
            self.turtles += exploration_step
            # --- Exploitation: Calculate velocity change --
            new_velocities = np.zeros((self.num_turtles, self.
               dimensions))
            for i in range(self.num_turtles):
             temp_factor = np.float64(self.temperature_impact(
                self.sand_temperature[i]))
                time_factor = np.float64(self.time_of_day_impact(
                   self.time_of_day[i]))
                efk=np.random.rand()
                emergence_factor = np.float64(self.
                   emergence_impact(self.emergence_types[i])*efk)
                if temp_factor == -np.inf:  # Turtle dies, remove
                    it
                    fitness_values[i] = np.inf
                    continue
                # Calculate change in velocity for each factor
```

```
                    separately
                velocity_change_temp = np.float64(np.float64(self
                    .velocities[i]) * np.float64(temp_factor))
                velocity_change_time = np.float64(np.float64(self
                    .velocities[i]) * np.float64(time_factor))
                velocity_change_emergence = emergence_factor
                # Sum up to get the final velocity change
                new_velocities[i] = np.float64(
                    velocity_change_temp + velocity_change_time +
                    velocity_change_emergence)
            # Update positions based on new velocities
            self.velocities = new_velocities
            self.turtles += self.velocities
            # Ensure turtles stay within bounds
            self.turtles = np.clip(self.turtles, self.lower_bound
                , self.upper_bound)
            if ((iteration+1)%100==0):
                print(f"Iteration {iteration+1}, Best Fitness: {
                    self.best_fitness}")
        return self.best_turtle, self.best_fitness
    def sphere_function(x):
        return np.sum(x**2)
```

3.2 Example Usage of Version 1

```
for i in range(5):
    orso = OliveRidleySurvivalOptimizer(objective_function=
        sphere_function, dimensions=5, max_iterations=500)
    best_solution, best_fitness = orso.optimize()
    print("\nBest Solution:", best_solution)
    print("Best Fitness:", best_fitness)
    print()
```

3.3 Output (Sample) of Version 1

```
Iteration 100, Best Fitness: 2.3662830162697004
 Iteration 200, Best Fitness: 2.3662830162697004
 Iteration 300, Best Fitness: 2.3662830162697004
 Iteration 400, Best Fitness: 2.3662830162697004
 Iteration 500, Best Fitness: 2.3662830162697004
 Best Solution: [ 0.65992088 -0.00950264  0.71581274  0.31473999 -1.14858522]
 Best Fitness: 2.3662830162697004
 Iteration 100, Best Fitness: 6.1582434543872475
 Iteration 200, Best Fitness: 6.1582434543872475
 Iteration 300, Best Fitness: 6.1582434543872475
 Iteration 400, Best Fitness: 6.1582434543872475
 Iteration 500, Best Fitness: 6.1582434543872475
 Best Solution: [ 0.59660102 -0.44951186 -0.8119812  -1.10835194 -1.92678288]
```

```
Best Fitness: 6.1582434543872475
Iteration 100, Best Fitness: 7.609961821346505
Iteration 200, Best Fitness: 7.609961821346505
Iteration 300, Best Fitness: 7.609961821346505
Iteration 400, Best Fitness: 7.609961821346505
Iteration 500, Best Fitness: 7.609961821346505
Best Solution: [-1.2811897  -0.99473339 -1.60118799 -1.51810149 -0.33254343]
Best Fitness: 7.609961821346505
Iteration 100, Best Fitness: 7.2877166059454375
Iteration 200, Best Fitness: 7.2877166059454375
Iteration 300, Best Fitness: 7.2877166059454375
Iteration 400, Best Fitness: 7.2877166059454375
Iteration 500, Best Fitness: 7.2877166059454375
Best Solution: [-0.51924519 -0.90285555 -1.8556707   1.49460903  0.72497103]
Best Fitness: 7.2877166059454375
Iteration 100, Best Fitness: 8.493423652317485
Iteration 200, Best Fitness: 4.156050083536117
Iteration 300, Best Fitness: 4.156050083536117
Iteration 400, Best Fitness: 4.156050083536117
Iteration 500, Best Fitness: 4.156050083536117
Best Solution: [-0.12672736  1.32018604  1.45483458  0.23623694  0.47407546]
Best Fitness: 4.156050083536117
```

3.4 Variation 2 : Original but with adaptive learning rate.

```python
class ModifiedOliveRidleySurvivalOptimizer_2:
    def __init__(self, objective_function, num_turtles=30,
       dimensions=2, lower_bound=-100, upper_bound=100,
                 max_iterations=100, tolerable_temp=28, max_temp
                    =35, time_range=(6, 18), beta=0.5,
                    learning_rate=0.1):
        self.obj_func = objective_function
        self.num_turtles = num_turtles
        self.dimensions = dimensions
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.max_iterations = max_iterations

        # Adaptive Beta
        self.beta = beta
        self.learning_rate = learning_rate

        # Initialize turtle positions and velocities
        self.turtles = np.random.uniform(lower_bound, upper_bound
           , (num_turtles, dimensions)).astype(np.float64)
        self.velocities = np.random.uniform(-1, 1, (num_turtles,
           dimensions)).astype(np.float64)

        self.best_turtle = None
```

```
        self.best_fitness = float("inf")

        # Biological parameters
        self.sand_temperature = np.random.uniform(25, 38,
           num_turtles).astype(np.float64)
        self.time_of_day = np.random.uniform(0, 24, num_turtles).
           astype(np.float64)
        self.emergence_types = np.random.choice([-1, 0, 1],
           num_turtles).astype(np.float64)

        self.tolerable_temp = tolerable_temp
        self.max_temp = max_temp
        self.time_range = time_range

    def temperature_impact(self, temp):
        if temp < self.tolerable_temp:
            return 1 + np.exp(-0.1 * (self.tolerable_temp - temp)
               )
        elif self.tolerable_temp <= temp <= self.max_temp:
            return np.exp(-0.1 * (temp - self.tolerable_temp))
        else:
            return -np.inf

    def time_of_day_impact(self, time):
        return 1.5 if self.time_range[0] <= time <= self.
           time_range[1] else 0.5

    def emergence_impact(self, emergence_type):
        return emergence_type
```

```
    def optimize(self):
        convergence=[]
        for iteration in range(self.max_iterations):
            fitness_values = np.apply_along_axis(self.obj_func,
               1, self.turtles)

            min_index = np.argmin(fitness_values)
            if fitness_values[min_index] < self.best_fitness:
                self.best_fitness = fitness_values[min_index]
                self.best_turtle = self.turtles[min_index].copy()
            convergence.append(self.best_fitness)

            random_factor = np.random.uniform(0, 1, (self.
               num_turtles, self.dimensions))
            movement_direction = np.sign(np.random.uniform(-1, 1,
                (self.num_turtles, self.dimensions)))
            exploration_step = (1 - self.beta) *
               movement_direction * random_factor
            self.turtles += exploration_step

            new_velocities = np.zeros((self.num_turtles, self.
               dimensions))
```

```
            for i in range(self.num_turtles):
                temp_factor = self.temperature_impact(self.
                    sand_temperature[i])
                time_factor = self.time_of_day_impact(self.
                    time_of_day[i])
                emergence_factor = self.emergence_impact(self.
                    emergence_types[i])

                if temp_factor == -np.inf:
                    fitness_values[i] = np.inf
                    continue

                velocity_change_temp = self.velocities[i] *
                    temp_factor
                velocity_change_time = self.velocities[i] *
                    time_factor
                velocity_change_emergence = self.velocities[i] *
                    emergence_factor

                #new_velocities[i] = (self.beta * (
                    velocity_change_temp + velocity_change_time) +
                     (0.5+0.5*(1 - self.beta)) *
                    velocity_change_emergence)
                new_velocities[i] = (self.beta * (
                    velocity_change_temp + velocity_change_time) +
                     (self.beta) * velocity_change_emergence)

            self.velocities = new_velocities
            self.turtles += self.velocities
            self.turtles = np.clip(self.turtles, self.lower_bound
                , self.upper_bound)

            improvement_rate = abs(self.best_fitness - np.min(
                fitness_values)) / (abs(self.best_fitness) + 1e-8)
            self.beta = max(0.1, min(0.9, self.beta * (1 - self.
                learning_rate * improvement_rate)))

            if (iteration + 1) % 100 == 0:
                print(f"Iteration {iteration + 1}, Best Fitness:
                    {self.best_fitness}, Beta: {self.beta:.2f}")

        return self.best_turtle, self.best_fitness,convergence


def sphere_function(x):
    return np.sum(x**2)  # Sphere function (min at x = 0)
```

3.5 Example Usage of Version 2

```
convfull=[]
for i in range(5):
  orso = ModifiedOliveRidleySurvivalOptimizer_2(
     objective_function=sphere_function, dimensions=5,
     max_iterations=500)
  best_solution, best_fitness,convergence = orso.optimize()
  print("\nBest Solution:", best_solution)
  print("Best Fitness:", best_fitness)
  convfull.append(convergence)
  print()
```

3.6 Output (Sample) of Version 2

```
Iteration 100, Best Fitness: 5880.335698922154, Beta: 0.10
Iteration 200, Best Fitness: 5880.335698922154, Beta: 0.10
Iteration 300, Best Fitness: 4197.774893098292, Beta: 0.10
Iteration 400, Best Fitness: 4197.774893098292, Beta: 0.10
Iteration 500, Best Fitness: 4197.774893098292, Beta: 0.10

Best Solution: [ 18.8161305  -45.58823223  34.65811058 -20.60848979 -11.81298981]
Best Fitness: 4197.774893098292

Iteration 100, Best Fitness: 4993.64689638618, Beta: 0.10
Iteration 200, Best Fitness: 4068.579595055809, Beta: 0.10
Iteration 300, Best Fitness: 3325.1265608911376, Beta: 0.10
Iteration 400, Best Fitness: 3325.1265608911376, Beta: 0.10
Iteration 500, Best Fitness: 3325.1265608911376, Beta: 0.10

Best Solution: [ 31.77704704 -41.61102163   5.65069713  13.3812792   19.31009348]
Best Fitness: 3325.1265608911376

Iteration 100, Best Fitness: 3507.003661367452, Beta: 0.10
Iteration 200, Best Fitness: 3507.003661367452, Beta: 0.10
Iteration 300, Best Fitness: 3507.003661367452, Beta: 0.10
Iteration 400, Best Fitness: 3507.003661367452, Beta: 0.10
Iteration 500, Best Fitness: 3507.003661367452, Beta: 0.10

Best Solution: [-36.90698687  -2.38891634  28.26467419 -35.47827525  -9.0316797 ]
Best Fitness: 3507.003661367452

Iteration 100, Best Fitness: 460.27753703761425, Beta: 0.10
Iteration 200, Best Fitness: 460.27753703761425, Beta: 0.10
Iteration 300, Best Fitness: 460.27753703761425, Beta: 0.10
Iteration 400, Best Fitness: 460.27753703761425, Beta: 0.10
Iteration 500, Best Fitness: 460.27753703761425, Beta: 0.10

Best Solution: [  5.87305636  -3.28454561   6.29107675 -13.04958199 -14.32226481]
```

```
Best Fitness: 460.27753703761425

Iteration 100, Best Fitness: 4095.7635873705203, Beta: 0.30
Iteration 200, Best Fitness: 3879.818829915866, Beta: 0.20
Iteration 300, Best Fitness: 3538.9339869270057, Beta: 0.11
Iteration 400, Best Fitness: 3428.742005158181, Beta: 0.10
Iteration 500, Best Fitness: 3315.022069670792, Beta: 0.10

Best Solution: [-33.12247317 -29.59404641   5.38613056 -30.41121307  19.70441511]
Best Fitness: 3315.022069670792
```

3.7 Variation 3: Realistic exploration phase but without global best and adaptive velocity use

```
class ModifiedOliveRidleySurvivalOptimizer_3:
    def __init__(self, objective_function, num_turtles=30,
       dimensions=2, lower_bound=-100, upper_bound=100,
                 max_iterations=100, tolerable_temp=28, max_temp
                    =35, time_range=(6, 18)):
        self.obj_func = objective_function
        self.num_turtles = num_turtles
        self.dimensions = dimensions
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.max_iterations = max_iterations

        # Positions
        self.turtles = np.random.uniform(lower_bound, upper_bound
           , (num_turtles, dimensions))
        self.best_turtle = None
        self.best_fitness = float("inf")

        # Biological Factors
        self.sand_temperature = np.random.uniform(25, 38,
           num_turtles)
        self.time_of_day = np.random.uniform(0, 24, num_turtles)
        self.emergence_types = np.random.choice([-1, 0, 1],
           num_turtles)

        # Constants
        self.tolerable_temp = tolerable_temp
        self.max_temp = max_temp
        self.time_range = time_range

    def temperature_impact(self, temp):
        if temp < self.tolerable_temp:
            return 1 + np.exp(-0.1 * (self.tolerable_temp - temp)
               )
        elif self.tolerable_temp <= temp <= self.max_temp:
            return np.exp(-0.1 * (temp - self.tolerable_temp))
        else:
            return -np.inf

    def time_of_day_impact(self, time):
        return 1.5 if self.time_range[0] <= time <= self.
           time_range[1] else 0.5

    def emergence_impact(self, emergence_type):
        return emergence_type
```

```
    def optimize(self):
        for iteration in range(self.max_iterations):
            fitness_values = np.apply_along_axis(self.obj_func,
```

```
                1, self.turtles)
            min_idx = np.argmin(fitness_values)
            if fitness_values[min_idx] < self.best_fitness:
                self.best_fitness = fitness_values[min_idx]
                self.best_turtle = self.turtles[min_idx].copy()

            # Exploration: biologically driven position update
            for i in range(self.num_turtles):
                temp_factor = self.temperature_impact(self.
                    sand_temperature[i])
                if temp_factor == -np.inf:
                    continue  # Dead turtle

                time_factor = self.time_of_day_impact(self.
                    time_of_day[i])
                emergence_factor = self.emergence_impact(self.
                    emergence_types[i]) * np.random.rand()

                # Biological movement
                random_walk = np.random.uniform(-1, 1, self.
                    dimensions)
                total_biological_effect = (temp_factor +
                    time_factor + emergence_factor) * random_walk
                self.turtles[i] += total_biological_effect

            self.turtles = np.clip(self.turtles, self.lower_bound
                , self.upper_bound)

        return self.best_turtle, self.best_fitness

def sphere(x): return np.sum(x**2)
```

3.8 Example Usage of Version 3

```
orso_bio = ModifiedOliveRidleySurvivalOptimizer_3(
   objective_function=sphere, dimensions=5, max_iterations=500)
sol, fit = orso_bio.optimize()
print("Best:", sol, "\nFitness:", fit)
```

3.9 Output (Sample) of Version 3

```
Best: [-37.08506937 -15.84530931 -11.08911885  32.85061605  38.83940591]
Fitness: 4337.007180962279
```

3.10 Variation 4: Bio hybrid ORS but temperature effect is linear

```
class ModifiedOliveRidleySurvivalOptimizer_4:
    def __init__(self, objective_function, num_turtles=30,
       dimensions=2, lower_bound=-100, upper_bound=100,
                 max_iterations=100, tolerable_temp=28, max_temp
                    =35, time_range=(6, 18), beta=0.5,
                    learning_rate=0.1):
        self.obj_func = objective_function
        self.num_turtles = num_turtles
        self.dimensions = dimensions
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.max_iterations = max_iterations

        self.beta = beta
        self.learning_rate = learning_rate

        self.turtles = np.random.uniform(lower_bound, upper_bound
           , (num_turtles, dimensions))
        self.velocities = np.random.uniform(-1, 1, (num_turtles,
           dimensions))

        self.best_turtle = None
        self.best_fitness = float("inf")

        self.sand_temperature = np.random.uniform(25, 38,
           num_turtles)
        self.time_of_day = np.random.uniform(0, 24, num_turtles)
        self.emergence_types = np.random.choice([-1, 0, 1],
           num_turtles)

        self.tolerable_temp = tolerable_temp
        self.max_temp = max_temp
        self.time_range = time_range

    def temperature_impact(self, temp):
        if temp < self.tolerable_temp:
            return 1 + (self.tolerable_temp - temp) / self.
               tolerable_temp
        elif self.tolerable_temp <= temp <= self.max_temp:
            return 1 - (temp - self.tolerable_temp) / (self.
               max_temp - self.tolerable_temp)
        else:
            return -np.inf

    def time_of_day_impact(self, time):
        return 1.5 if self.time_range[0] <= time <= self.
           time_range[1] else 0.5
```

```
    def emergence_impact(self, emergence_type):
        return emergence_type
```

```
    def optimize(self):
        convergence = []

        for iteration in range(self.max_iterations):
            fitness_values = np.apply_along_axis(self.obj_func,
               1, self.turtles)

            min_index = np.argmin(fitness_values)
            if fitness_values[min_index] < self.best_fitness:
                self.best_fitness = fitness_values[min_index]
                self.best_turtle = self.turtles[min_index].copy()
            convergence.append(self.best_fitness)

            exploration_noise = np.random.uniform(-1, 1, (self.
               num_turtles, self.dimensions))
            exploration_step = (1 - self.beta) *
               exploration_noise
            self.turtles += exploration_step

            new_velocities = np.zeros_like(self.velocities)

            for i in range(self.num_turtles):
                temp_factor = self.temperature_impact(self.
                   sand_temperature[i])
                time_factor = self.time_of_day_impact(self.
                   time_of_day[i])
                emergence_factor = self.emergence_impact(self.
                   emergence_types[i])

                if temp_factor == -np.inf:
                    fitness_values[i] = np.inf
                    continue

                velocity_change_temp = self.velocities[i] *
                   temp_factor
                velocity_change_time = self.velocities[i] *
                   time_factor
                velocity_change_emergence = self.velocities[i] *
                   emergence_factor

                new_velocities[i] = (self.beta * (
                   velocity_change_temp + velocity_change_time) +
                                     self.beta *
                                        velocity_change_emergence
                                        )

            self.velocities = new_velocities
            self.turtles += self.velocities
            self.turtles = np.clip(self.turtles, self.lower_bound
```

```
                , self.upper_bound)

            improvement_rate = abs(self.best_fitness - np.min(
                fitness_values)) / (abs(self.best_fitness) + 1e-8)
            self.beta = max(0.1, min(0.9, self.beta * (1 - self.
                learning_rate * improvement_rate)))

            if (iteration + 1) % 100 == 0:
                print(f"Iteration {iteration + 1}, Best Fitness:
                    {self.best_fitness:.4f}, Beta: {self.beta:.2f}
                    ")

        return self.best_turtle, self.best_fitness, convergence
```

3.11 Example Usage of Version 4

```
def sphere(x):
    return np.sum(x**2)

conv_all = []
for i in range(3):  # Fewer runs for testing
    orso = ModifiedOliveRidleySurvivalOptimizer_4(
        objective_function=sphere, dimensions=5, max_iterations
        =300)
    sol, fitness, conv = orso.optimize()
    print(f"\nRun {i+1}: Best Fitness = {fitness:.4f}")
    conv_all.append(conv)
```

3.12 Output (Sample) of Version 4

```
Iteration 100, Best Fitness: 5445.1244, Beta: 0.32
Iteration 200, Best Fitness: 4997.9553, Beta: 0.23
Iteration 300, Best Fitness: 4284.4537, Beta: 0.16

Run 1: Best Fitness = 4284.4537
Iteration 100, Best Fitness: 3538.1491, Beta: 0.10
Iteration 200, Best Fitness: 3538.1491, Beta: 0.10
Iteration 300, Best Fitness: 3538.1491, Beta: 0.10

Run 2: Best Fitness = 3538.1491
Iteration 100, Best Fitness: 4096.2952, Beta: 0.10
Iteration 200, Best Fitness: 4096.2952, Beta: 0.10
Iteration 300, Best Fitness: 3995.3651, Beta: 0.10

Run 3: Best Fitness = 3995.3651
```

3.13 Variation 5: Bio hybrid ORS but emergence effect is same as original (-1,0,1)

```
class ModifiedOliveRidleySurvivalOptimizer_5:
    def __init__(self, objective_function, num_turtles=30,
       dimensions=2, lower_bound=-100, upper_bound=100,
                 max_iterations=100, tolerable_temp=28, max_temp
                    =35, time_range=(6, 18), beta=0.5,
                    learning_rate=0.1):
        self.obj_func = objective_function
        self.num_turtles = num_turtles
        self.dimensions = dimensions
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.max_iterations = max_iterations

        self.beta = beta
        self.learning_rate = learning_rate

        self.turtles = np.random.uniform(lower_bound, upper_bound
           , (num_turtles, dimensions))
        self.velocities = np.random.uniform(-1, 1, (num_turtles,
           dimensions))

        self.best_turtle = None
        self.best_fitness = float("inf")

        self.sand_temperature = np.random.uniform(25, 38,
           num_turtles)
        self.time_of_day = np.random.uniform(0, 24, num_turtles)
        self.emergence_times = np.sort(np.random.rand(num_turtles
           ))  # Simulated emergence timestamps

        self.tolerable_temp = tolerable_temp
        self.max_temp = max_temp
        self.time_range = time_range

    def temperature_impact(self, temp):
        if temp < self.tolerable_temp:
            return np.exp((self.tolerable_temp - temp) / self.
               tolerable_temp)
        elif self.tolerable_temp <= temp <= self.max_temp:
            return np.exp(-(temp - self.tolerable_temp) / (self.
               max_temp - self.tolerable_temp + 1e-8))
        else:
            return -np.inf

    def time_of_day_impact(self, time):
        return 1.5 if self.time_range[0] <= time <= self.
           time_range[1] else 0.5
```

```
    def emergence_impact(self, emergence_rank):
        # Normalize between -1 (latest) and +1 (earliest)
        return 1.0 - 2.0 * emergence_rank  # earliest = 0     +1,
            latest = 1     -1
```

```
    def optimize(self):
        convergence = []

        for iteration in range(self.max_iterations):
            fitness_values = np.apply_along_axis(self.obj_func,
               1, self.turtles)

            min_index = np.argmin(fitness_values)
            if fitness_values[min_index] < self.best_fitness:
                self.best_fitness = fitness_values[min_index]
                self.best_turtle = self.turtles[min_index].copy()
            convergence.append(self.best_fitness)

            exploration_noise = np.random.uniform(-1, 1, (self.
               num_turtles, self.dimensions))
            exploration_step = (1 - self.beta) *
               exploration_noise
            self.turtles += exploration_step

            new_velocities = np.zeros_like(self.velocities)

            emergence_ranks = np.argsort(np.argsort(self.
               emergence_times)) / (self.num_turtles - 1)

            for i in range(self.num_turtles):
                temp_factor = self.temperature_impact(self.
                   sand_temperature[i])
                time_factor = self.time_of_day_impact(self.
                   time_of_day[i])
                emergence_factor = self.emergence_impact(
                   emergence_ranks[i])

                if temp_factor == -np.inf:
                    fitness_values[i] = np.inf
                    continue

                velocity_change_temp = self.velocities[i] *
                   temp_factor
                velocity_change_time = self.velocities[i] *
                   time_factor
                velocity_change_emergence = self.velocities[i] *
                   emergence_factor

                new_velocities[i] = (self.beta * (
                   velocity_change_temp + velocity_change_time) +
                                     self.beta *
                                        velocity_change_emergence
```

```
                                        )

            self.velocities = new_velocities
            self.turtles += self.velocities
            self.turtles = np.clip(self.turtles, self.lower_bound
                , self.upper_bound)

            improvement_rate = abs(self.best_fitness - np.min(
                fitness_values)) / (abs(self.best_fitness) + 1e-8)
            self.beta = max(0.1, min(0.9, self.beta * (1 - self.
                learning_rate * improvement_rate)))

            if (iteration + 1) % 100 == 0:
                print(f"Iteration {iteration + 1}, Best Fitness:
                    {self.best_fitness:.4f}, Beta: {self.beta:.2f}
                    ")

        return self.best_turtle, self.best_fitness, convergence
```

3.14 Example Usage of Version 5

```
def sphere(x):
    return np.sum(x**2)

conv_all = []
for i in range(3):
    orso = ModifiedOliveRidleySurvivalOptimizer_5(
        objective_function=sphere, dimensions=5, max_iterations
        =300)
    sol, fitness, conv = orso.optimize()
    print(f"\nRun {i+1}: Best Fitness = {fitness:.4f}")
    conv_all.append(conv)
```

3.15 Output (Sample) of Version 5

```
Iteration 100, Best Fitness: 3401.2935, Beta: 0.10
Iteration 200, Best Fitness: 3401.2935, Beta: 0.10
Iteration 300, Best Fitness: 3401.2935, Beta: 0.10

Run 1: Best Fitness = 3401.2935
Iteration 100, Best Fitness: 1787.3401, Beta: 0.10
Iteration 200, Best Fitness: 1787.3401, Beta: 0.10
Iteration 300, Best Fitness: 1787.3401, Beta: 0.10

Run 2: Best Fitness = 1787.3401
Iteration 100, Best Fitness: 1435.6062, Beta: 0.10
Iteration 200, Best Fitness: 1435.6062, Beta: 0.10
Iteration 300, Best Fitness: 1435.6062, Beta: 0.10
```

```
Run 3: Best Fitness = 1435.6062
```

4 CEC Benchmarks Tables and Graphs

4.1 Optimization Algorithms used for comparison

Table 1: Full Forms of Optimization Algorithms

Abbreviation	Full Form
ORG	Original Olive Ridley Survival Optimizer
MOD	Modified Olive Ridley Survival Optimizer
SCA	Sine Cosine Algorithm
ChOA	Chimp Optimization Algorithm
SSA	Salp Swarm Algorithm
GWO	Grey Wolf Optimizer
WOA	Whale Optimization Algorithm
AOA	Arithmetic Optimization Algorithm
PSO	Particle Swarm Optimization
BA	Bat Algorithm
MFO	Moth Flame Optimization
DE	Differential Evolution

4.2 CEC-2014 Tables

These tables give the values of the mean and standard deviation of the 30 CEC 2014 benchmark functions.

Table 2: CEC-2014 Mean Values

ORG	MOD	SCA	ChOA	SSA	GWO	WOA	AOA
3.61E+09	4.39E+06	4.19E+08	5.99E+08	2.38E+07	8.94E+07	8.55E+07	1.36E+09
1.45E+11	8.91E+03	2.65E+10	4.38E+10	1.30E+04	2.46E+09	3.43E+09	7.13E+10
5.11E+05	5.18E+04	5.88E+04	8.27E+04	7.37E+04	4.53E+04	4.59E+04	8.10E+04
4.31E+04	4.97E+02	2.55E+03	3.34E+03	5.32E+02	6.97E+02	6.62E+02	1.23E+04
5.21E+02	5.20E+02	5.21E+02	5.21E+02	5.20E+02	5.21E+02	5.21E+02	5.21E+02
6.48E+02	6.30E+02	6.37E+02	6.36E+02	6.23E+02	6.16E+02	6.16E+02	6.39E+02
1.96E+03	7.00E+02	9.17E+02	1.18E+03	7.02E+02	7.23E+02	7.23E+02	1.35E+03
1.30E+03	9.61E+02	1.07E+03	1.07E+03	9.57E+02	8.88E+02	8.89E+02	1.15E+03
1.53E+03	1.08E+03	1.21E+03	1.19E+03	1.06E+03	1.01E+03	1.02E+03	1.22E+03
9.44E+03	4.89E+03	7.67E+03	7.80E+03	4.78E+03	3.60E+03	3.56E+03	7.27E+03
9.72E+03	5.28E+03	8.68E+03	8.94E+03	5.05E+03	4.27E+03	4.48E+03	7.72E+03
1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03	1.20E+03
1.31E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.30E+03	1.31E+03
1.85E+03	1.40E+03	1.47E+03	1.56E+03	1.40E+03	1.41E+03	1.41E+03	1.64E+03
2.78E+07	1.52E+03	1.85E+04	1.26E+05	1.51E+03	1.80E+03	1.80E+03	2.96E+05
1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03	1.61E+03
3.91E+08	2.03E+05	1.39E+07	3.29E+07	1.27E+06	2.75E+06	2.72E+06	8.81E+07
1.24E+10	1.16E+05	3.13E+08	9.12E+08	1.09E+04	2.20E+07	1.69E+07	2.34E+09
5.89E+09	2.61E+03	2.03E+03	2.16E+03	1.92E+03	1.95E+03	1.97E+03	2.25E+03
2.30E+16	1.07E+05	4.45E+04	1.08E+05	3.28E+04	3.06E+04	2.49E+04	2.21E+05
4.45E+09	1.41E+05	4.07E+06	1.13E+07	4.14E+05	8.92E+05	1.33E+06	2.43E+07
5.95E+14	2.86E+03	3.26E+03	3.18E+03	2.79E+03	2.67E+03	2.61E+03	4.97E+03
4.11E+03	2.53E+03	2.72E+03	2.75E+03	2.64E+03	2.64E+03	2.64E+03	2.51E+03
3.01E+03	2.64E+03	2.61E+03	2.60E+03	2.64E+03	2.60E+03	2.60E+03	2.60E+03
2.96E+03	2.73E+03	2.74E+03	2.71E+03	2.72E+03	2.71E+03	2.71E+03	2.70E+03
2.84E+03	2.72E+03	2.70E+03	2.80E+03	2.70E+03	2.74E+03	2.74E+03	2.78E+03
4.58E+03	3.92E+03	3.86E+03	3.92E+03	3.57E+03	3.38E+03	3.38E+03	3.83E+03
9.31E+03	5.61E+03	5.59E+03	5.75E+03	4.27E+03	4.19E+03	4.21E+03	4.97E+03
4.72E+09	2.11E+06	3.03E+07	5.15E+07	4.55E+06	3.43E+06	1.25E+07	3.32E+08
1.63E+14	2.73E+06	4.65E+05	8.10E+05	3.88E+04	8.48E+04	8.64E+04	5.31E+06

Table 3: CEC-2014 Standard Deviation Values

ORG	MOD	SCA	ChOA	SSA	GWO	WOA	AOA
1160592709	2204010.915	1.09E+08	1.08E+08	1.17E+07	6.04E+07	5.42E+07	3.24E+08
20172192146	6979.867905	4.30E+09	6.87E+09	1.07E+04	2.54E+09	3.14E+09	1.12E+10
895496.8367	28252.45892	1.19E+04	8.02E+03	1.94E+04	1.22E+04	1.06E+04	8.24E+03
12193.52718	30.4912826	7.61E+02	1.43E+03	4.51E+01	1.22E+02	5.32E+01	3.53E+03
0.0647395	0.026004396	0.05832	0.05332	0.09926	0.05761	0.05325	0.06787
1.857486116	3.009186061	2.705	2.021	3.635	3.113	2.646	1.982
201.1568137	0.024099714	3.11E+01	8.54E+01	0.01456	1.90E+01	1.95E+01	1.11E+01
48.42595739	38.44271006	2.54E+01	2.21E+01	4.21E+01	2.44E+01	2.09E+01	3.07E+01
59.63221163	41.19630129	2.35E+01	2.15E+01	4.24E+01	2.04E+01	3.80E+01	2.79E+01
348.385532	789.2443259	4.63E+02	9.38E+02	7.85E+02	7.43E+02	5.14E+02	6.12E+02
392.2234259	740.0441565	3.16E+02	2.90E+02	7.85E+02	6.53E+02	1.22E+03	5.14E+02
0.478358215	0.065683341	0.3934	0.4052	0.4315	1.214	1.104	0.4557
1.417215507	0.098743874	0.2691	0.5682	0.1538	0.3385	0.5589	0.9561
67.5738678	0.157680765	1.12E+01	3.48E+01	2.232e1	8.419	9.241	3.68E+01
13539046.46	5.213228131	1.14E+04	9.52E+04	4.705	5.66E+02	7.87E+02	1.13E+05
0.219462215	0.508877513	0.2372	0.2239	0.7115	0.6959	0.7963	0.3641
190609859.5	101874.6199	7.72E+06	1.95E+07	8.69E+05	2.88E+06	2.55E+06	3.98E+07
4197102453	43622.89368	1.34E+08	1.00E+09	7.51E+03	4.06E+07	2.72E+07	1.63E+09
4180264609	1552.781151	3.65E+01	1.07E+02	1.78E+01	3.45E+01	4.45E+01	1.16E+02
3.4E+16	34608.52916	2.33E+04	3.92E+04	1.68E+04	1.27E+04	6.62E+04	8.62E+04
2682703526	98175.95739	2.34E+06	3.25E+06	3.89E+05	2.09E+06	2.13E+06	2.13E+07
7.68008E+14	298.1383149	2.17E+02	2.65E+02	2.09E+02	2.05E+02	2.76E+02	2.11E+03
676.8977693	3.860239549	1.95E+01	4.45E+01	1.23E+01	1.27E+01	1.04E+01	6.23E+01
49.58769167	13.54638623	1.95E+01	4.626e2	7.915	1.347e2	1.094e2	8.515e2
76.1046155	12.76807534	1.19E+01	1.37E+01	6.265	4.859	5.356	0
123.6860396	39.6562975	0.5079	4.84E+01	0.1473	5.97E+01	4.96E+01	3.37E+01
176.5576339	117.0284625	2.73E+02	2.00E+02	2.06E+02	1.29E+02	1.50E+02	5.43E+02
1367.878545	787.0937169	4.55E+02	2.48E+02	4.65E+02	4.41E+02	3.74E+02	2.70E+03
2215221688	6427746.957	1.49E+07	3.31E+07	7.90E+06	7.41E+06	8.59E+06	2.61E+08
2.96032E+14	5379243.299	1.78E+05	2.14E+05	2.15E+04	5.22E+04	5.85E+04	3.83E+06

Table 4: CEC-2014 Ranks

ORG	MOD	SCA	ChOA	SSA	GWO	WOA	AOA
8	1	5	6	2	4	3	7
8	1	5	6	2	3	4	7
8	3	4	7	5	1	2	6
8	1	5	6	2	4	3	7
5	1	6	6	2	4	6	3
8	4	6	5	3	2	1	7
8	1	5	6	2	3	4	7
8	4	6	5	3	1	2	7
8	4	6	5	3	1	2	7
8	4	6	7	3	2	1	5
8	4	6	7	3	1	2	5
8	1	5	5	2	3	5	3
8	1	5	5	2	2	2	7
8	1	5	6	2	3	3	7
8	2	5	6	1	3	4	7
8	3	4	4	4	1	1	4
8	1	5	6	2	4	3	7
8	2	5	6	1	4	3	7
8	7	4	5	1	2	3	6
8	5	4	6	3	2	1	7
8	1	5	6	2	3	4	7
8	4	6	5	3	2	1	7
8	2	6	7	3	4	5	1
8	7	5	1	6	1	1	1
8	6	7	2	5	2	4	1
8	3	2	7	1	4	5	6
8	6	5	7	3	1	2	4
8	6	5	7	3	1	2	4
8	1	5	6	3	2	4	7
8	6	4	5	1	2	3	7

4.3 CEC-2017 Tables

These tables give the values of the mean and standard deviation of the 28 CEC 2017 benchmark functions. Function 2 and 30 are not given as they have unstable convergence.

Table 5: CEC-2017 Mean Values

MOD	ORG	PSO	BA	GWO	WOA	MFO	SCA
5501.25	1.23E+11	1.41E+08	5.15E+05	2.05E+09	4.50E+06	1.35E+10	1.87E+10
358.28	4.95E+04	631.28	300.1	3.29E+04	1.39E+04	8.94E+04	3.81E+04
662.93	1.51E+05	479.01	440.77	671.09	579.96	1663	1486.8
500.00	500.10	691.03	738.84	585.81	689.31	715.14	753.3
767.16	5.06E+06	637.86	669.53	606.69	664.91	643.76	642.5
1089.52	1798.29	919.98	1677.8	877.24	1291.5	1178.4	1188.7
810.95	888.64	1050	1151.8	894.19	1088.7	1019.9	1067.1
5038.02	9384.36	6908.2	16166	2490.2	9602.2	8106.2	6761.2
48068.03	1.85E+10	5642.9	5513.4	3735.4	5856.2	5142.1	7892
3.33E+06	2.90E+10	1370.7	1466.1	3563.7	1470.5	9554.5	2870.8
136090.60	3.22E+10	9.13E+07	6.67E+06	2.47E+08	1.11E+08	1.06E+09	1.46E+09
242290.33	7.62E+07	2.71E+06	2.39E+05	1.37E+07	1.53E+05	3.56E+07	1.02E+08
97116.06	1.98E+10	29356	11166	180650	281830	147310	215250
6704.33	1.86E+10	289830	80184	3.50E+06	43017	46278	4.85E+06
2879.89	7.66E+15	2697.5	3270.9	2192.4	3221.1	3067.4	3356.6
204168.56	1.55E+08	2406.6	2829.9	2020.4	2600.7	2446.9	2517.7
70366.41	1.15E+16	75875	61472	5.78E+05	3.99E+06	5.25E+05	1.85E+06
3386.63	25782.83	333070	326870	365190	652720	9.43E+06	1.83E+07
2412.91	101011.48	2692.9	2967.8	2377.9	2691	2761.9	2723.6
3101.57	8592.91	2188.5	2159.5	2362.4	2265.5	3000.6	3070
2528.47	116044.45	2418.6	2481.6	2300.8	2423.8	2389.3	2445.3
2589.04	71826.23	4480.6	3489.8	2890.7	3147.6	2960.2	3280.9
2835.32	22252.89	2674.1	2734.5	3093	2707.5	3504.5	3852.2
3421.47	22177.30	2966.9	3015.4	3214.8	2710.6	3580.1	3634.5
3447.03	5098.20	3393.8	5274.2	5031.4	4215.6	6788.1	7925.8
2906.30	10470.99	5163.1	3972.1	3700.3	3934.3	3603.5	4030.1
31342.60	7.99E+14	3290.2	3331.1	3691.9	3125.7	5131.8	5697.4
359601.26	1.10E+14	4046.3	4714	3515.2	4324.4	4082.8	4278.9

Table 6: CEC-2017 Standard Deviation Values

MOD	ORG	PSO	BA	GWO	WOA	MFO	SCA
4660.45	1.73E+10	1.57E+07	2.42E+05	1.54E+09	3.98E+06	8.88E+09	3.29E+09
28.91	1.78E+04	39.18	0.10	1.01E+04	4.33E+03	6.40E+04	5.34E+03
123.78	2.03E+04	50.91	46.61	142.68	44.65	1082.20	271.30
0.0018	0.0233	23.35	51.93	25.37	46.13	47.74	14.00
50.15	6.55E+05	13.65	10.25	3.23	12.41	12.12	3.27
87.21	125.70	16.75	235.80	59.99	127.26	215.58	39.12
4.46	12.74	31.31	58.66	20.03	59.27	52.93	20.37
781.42	362.68	1979.90	5193.00	729.70	3407.10	2838.10	1358.10
2.11E+04	1.19E+10	508.86	621.38	623.12	814.14	852.33	298.77
3.12E+06	8.70E+09	58.99	162.49	4414.40	91.06	11425.00	364.56
1.32E+05	7.68E+09	4.03E+07	5.59E+06	6.45E+08	6.36E+07	1.86E+09	4.92E+08
1.07E+05	4.86E+07	7.92E+05	1.96E+05	2.42E+07	1.67E+05	1.35E+08	3.95E+07
3.78E+04	5.88E+09	17751.00	6816.40	4.28E+05	2.40E+05	376600.00	129610.00
8395.23	1.01E+10	101630.00	61904.00	1.34E+07	35044.00	47242.00	2.31E+06
592.99	1.89E+16	223.21	377.78	222.26	376.88	396.70	229.15
1.13E+05	9.39E+07	240.94	363.77	124.41	271.73	256.93	162.24
3.57E+04	1.40E+16	37712.00	35318.00	7.72E+05	3.65E+06	1.46E+06	1.24E+06
604.19	5470.12	2.01E+05	1.89E+05	1.66E+06	764410.00	4.74E+07	1.30E+07
219.85	2.11E+04	153.31	217.15	141.32	175.20	156.54	128.14
1007.17	1337.34	39.65	46.68	193.94	37.21	886.99	296.60
212.38	1.61E+04	40.22	79.43	28.64	42.84	44.01	43.18
154.28	8678.01	560.21	281.75	54.64	128.28	36.61	41.80
18.63	6117.65	38.89	384.16	352.07	330.32	32.05	243.41
97.64	1.09E+04	98.01	60.13	126.52	58.32	644.68	143.98
220.42	642.02	32.71	3663.70	1055.60	2617.50	459.67	386.63
176.48	2200.74	712.91	216.70	168.31	167.63	86.09	95.80
2.29E+04	1.06E+15	22.75	421.37	247.42	168.90	466.20	543.53
2.64E+05	1.24E+14	228.20	488.25	169.47	370.61	248.00	370.78

Table 7: CEC-2017 Ranks

MOD	ORG	PSO	BA	GWO	WOA	MFO	SCA
1	8	4	2	5	3	6	7
2	7	3	1	5	4	8	6
4	8	2	1	5	3	7	6
1	2	5	7	3	4	6	8
7	8	2	6	1	5	4	3
3	8	2	7	1	6	4	5
1	2	5	8	3	7	4	6
2	6	4	8	1	7	5	3
7	8	4	3	1	5	2	6
7	8	1	2	5	3	6	4
1	8	3	2	5	4	6	7
3	7	4	2	5	1	6	8
3	8	2	1	5	7	4	6
1	8	5	4	6	2	3	7
3	8	2	6	1	5	4	7
7	8	2	6	1	5	3	4
2	8	3	1	5	7	4	6
1	2	4	3	5	6	7	8
2	8	4	7	1	3	6	5
7	8	2	1	4	3	5	6
7	8	3	6	1	4	2	5
1	8	7	6	2	4	3	5
4	8	1	3	5	2	6	7
5	8	2	3	4	1	6	7
2	5	1	6	4	3	7	8
1	8	7	5	3	4	2	6
7	8	2	3	4	1	5	6
7	8	2	6	1	5	3	4

4.4 CEC-2020 Tables

These tables give the values of the mean and standard deviation of the 10 CEC 2020 benchmark functions.

Table 8: CEC-2020 Mean Values

MOD	ORG	AOA	BA	DE	GWO	PSO	WOA
3762.37	1.24E+11	6.21E+10	1.24E+11	100.00	1.02E+09	1.02E+08	4.05E+09
5578.14	9821.54	9703.47	10514.18	7206.76	4687.81	4546.91	7224.07
864.25	4.09E+06	2.09E+06	3.94E+06	860.27	35633.48	8670.57	2.27E+05
1928.79	1.47E+07	2.42E+06	9.59E+06	1912.28	1921.44	1950.96	7856.65
2.87E+05	6.44E+08	4.45E+08	3.82E+08	28371.37	5.04E+05	92001.76	1.19E+06
11281.51	4.43E+09	1.88E+09	1.74E+09	4167.12	34773.95	5313.37	22460.36
2.93E+05	5.87E+09	3.91E+09	2.14E+09	17184.23	7.07E+05	5.04E+05	1.29E+06
2876.90	8370.86	7034.68	5523.56	2372.04	2384.70	2494.12	3561.80
2705.71	6.92E+04	3.66E+04	5.16E+04	2623.21	4163.73	3517.83	11076.96
2939.60	2.17E+04	8743.53	19559.43	2920.88	3002.42	2957.55	3245.59

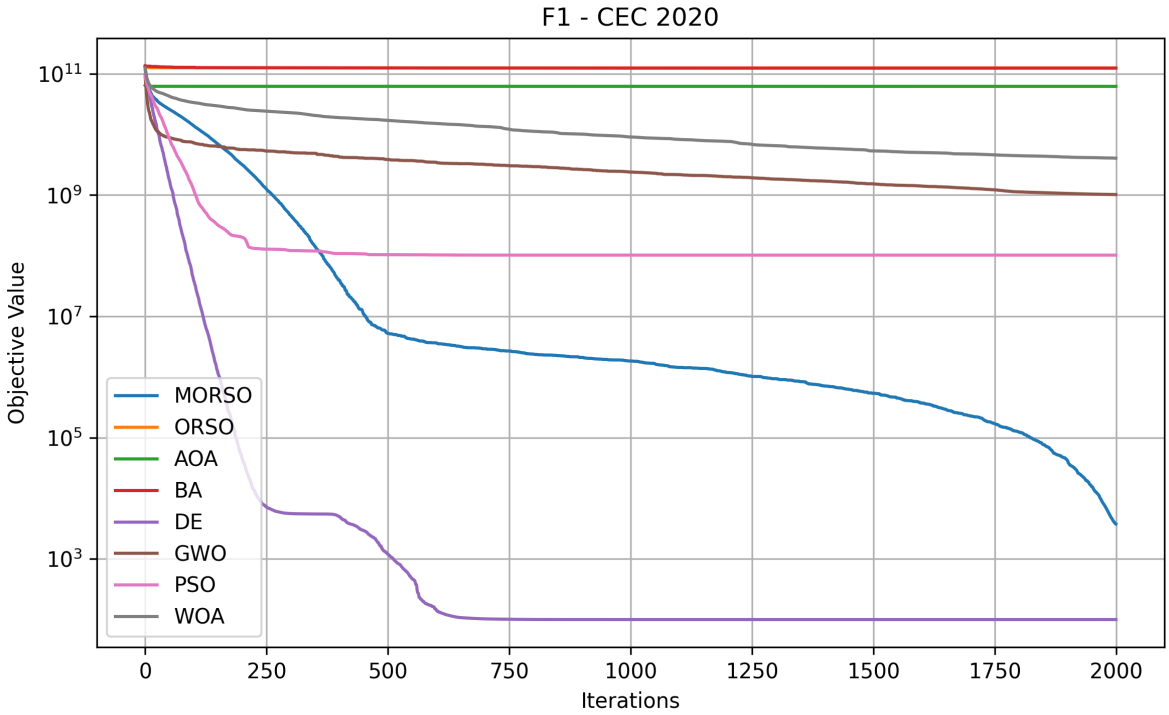
Table 9: CEC-2020 Standard Deviation Values

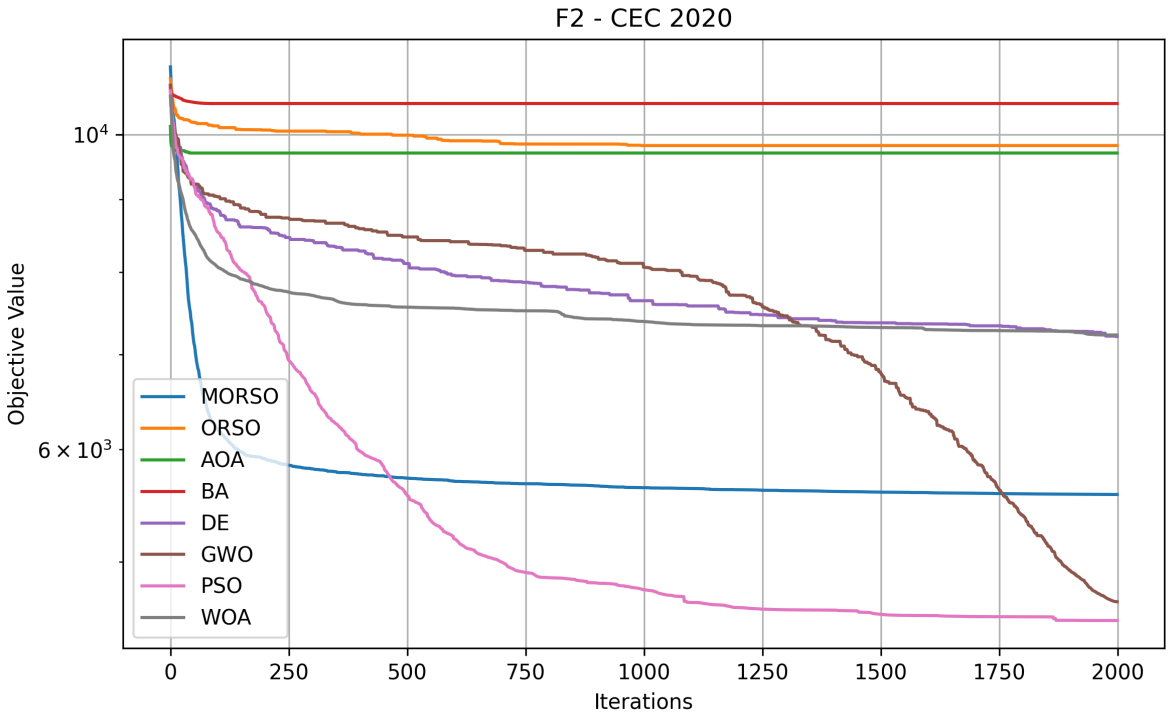
MOD	ORG	AOA	BA	DE	GWO	PSO	WOA
3056.04	2.01E+10	8.49E+09	1.82E+10	1.23E-11	1.10E+09	2.30E+08	4.89E+09
786.59	298.94	502.03	523.97	408.10	1599.22	947.89	1138.63
35.81	4.01E+05	1.86E+05	7.19E+05	14.57	3.07E+04	8413.85	9.35E+04
8.78	7.56E+06	1.02E+06	6.33E+06	1.20	13.18	27.62	3627.64
1.07E+05	3.66E+08	2.41E+08	2.48E+08	48172.84	3.36E+05	53790.27	1.69E+06
9942.33	2.32E+09	1.02E+09	1.30E+09	4903.53	15941.26	6964.66	18066.27
2.10E+05	3.87E+09	2.93E+09	1.08E+09	34036.46	3.34E+05	3.97E+05	1.89E+06
202.84	1270.83	887.75	1469.66	4.82	15.62	67.13	883.58
169.17	1.08E+04	1089.27	1.58E+04	70.00	1226.02	941.92	5066.18
33.32	5710.30	1352.52	4581.82	0.20	53.63	24.76	172.05

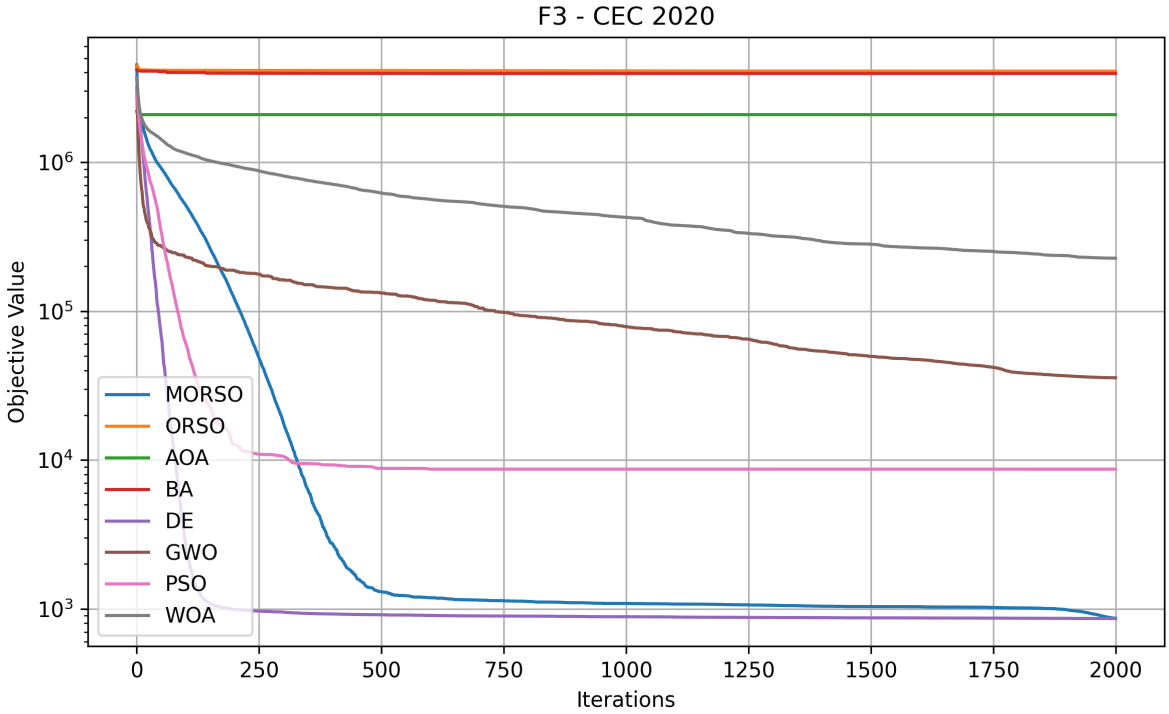
Table 10: CEC-2020 Ranks

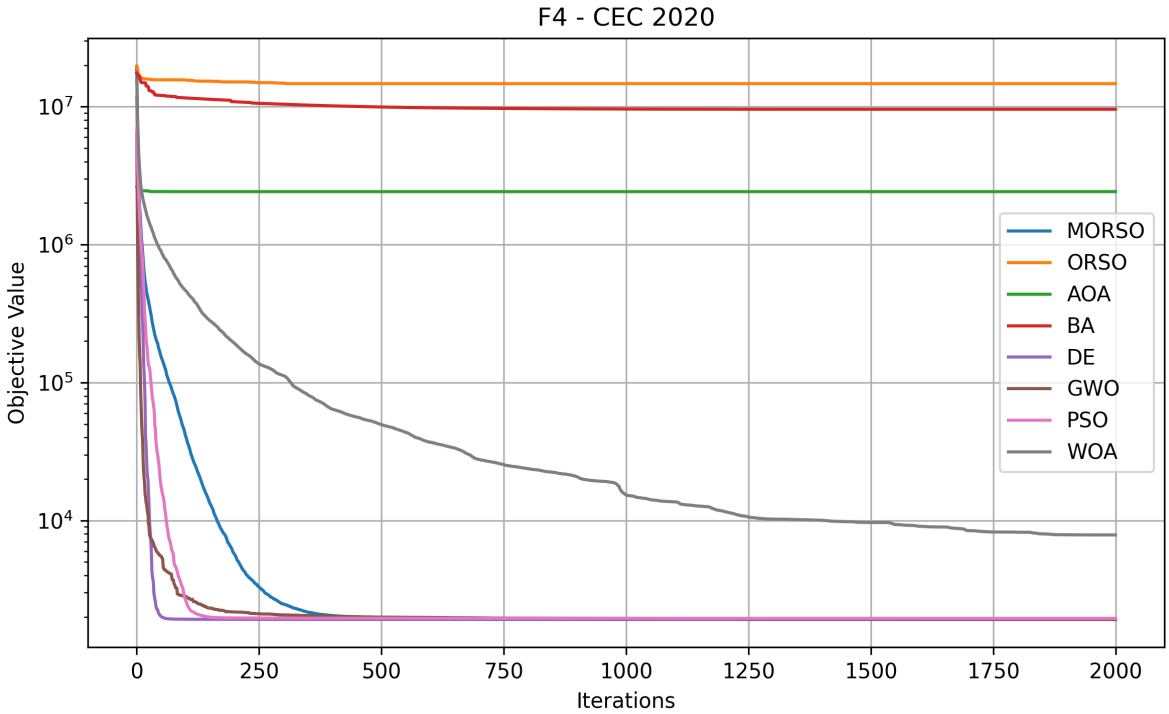
MOD	ORG	AOA	BA	DE	GWO	PSO	WOA
2	8	6	7	1	4	3	5
3	7	6	8	4	2	1	5
2	8	6	7	1	4	3	5
3	8	6	7	1	2	4	5
3	8	7	6	1	4	2	5
3	8	7	6	1	5	2	4
2	8	7	6	1	4	3	5
4	8	7	6	1	2	3	5
2	8	6	7	1	4	3	5
2	8	6	7	1	4	3	5

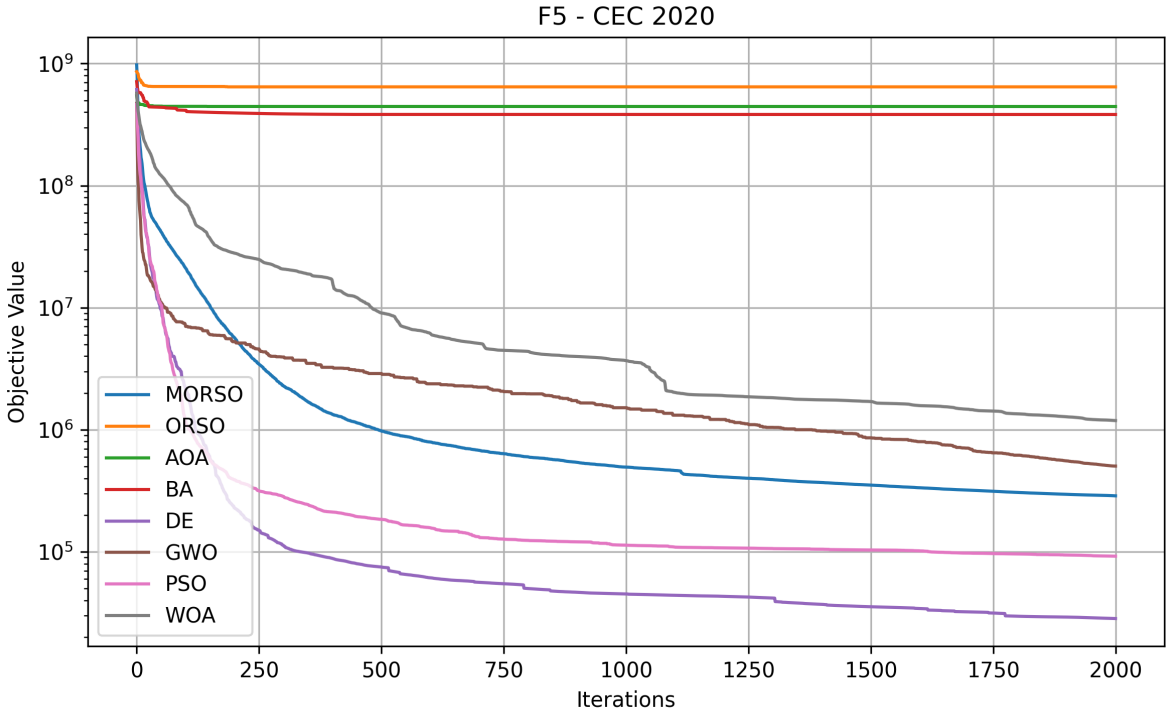
4.5 CEC 2020 Graphs

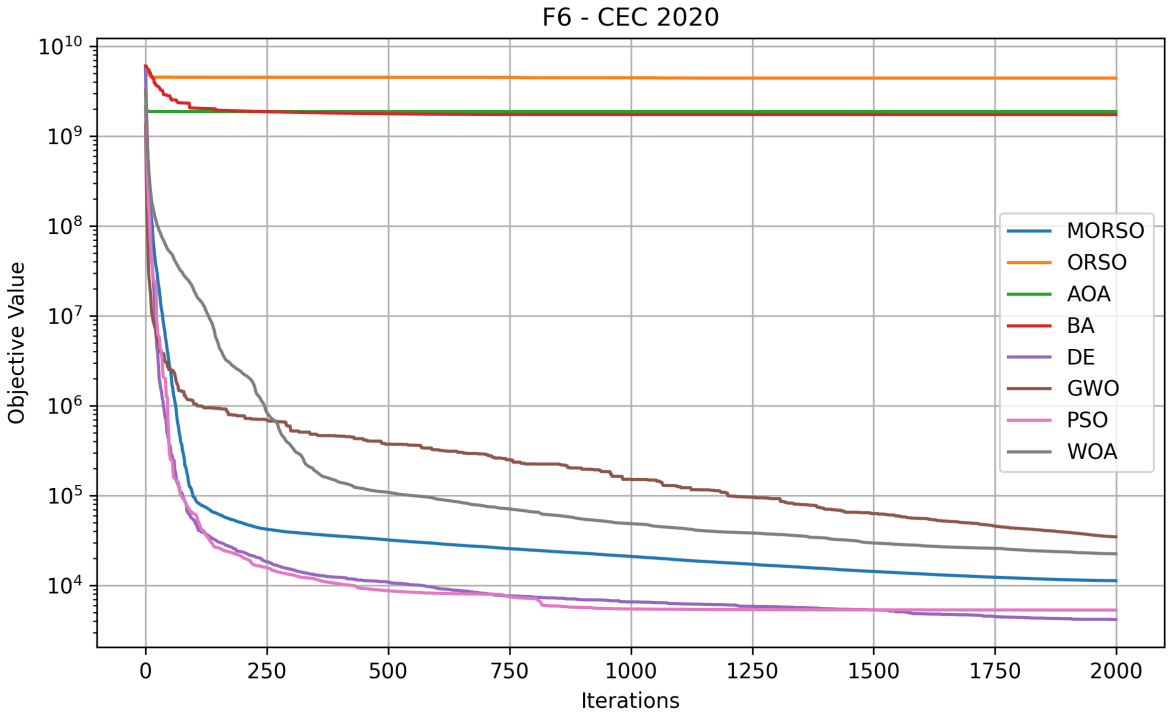


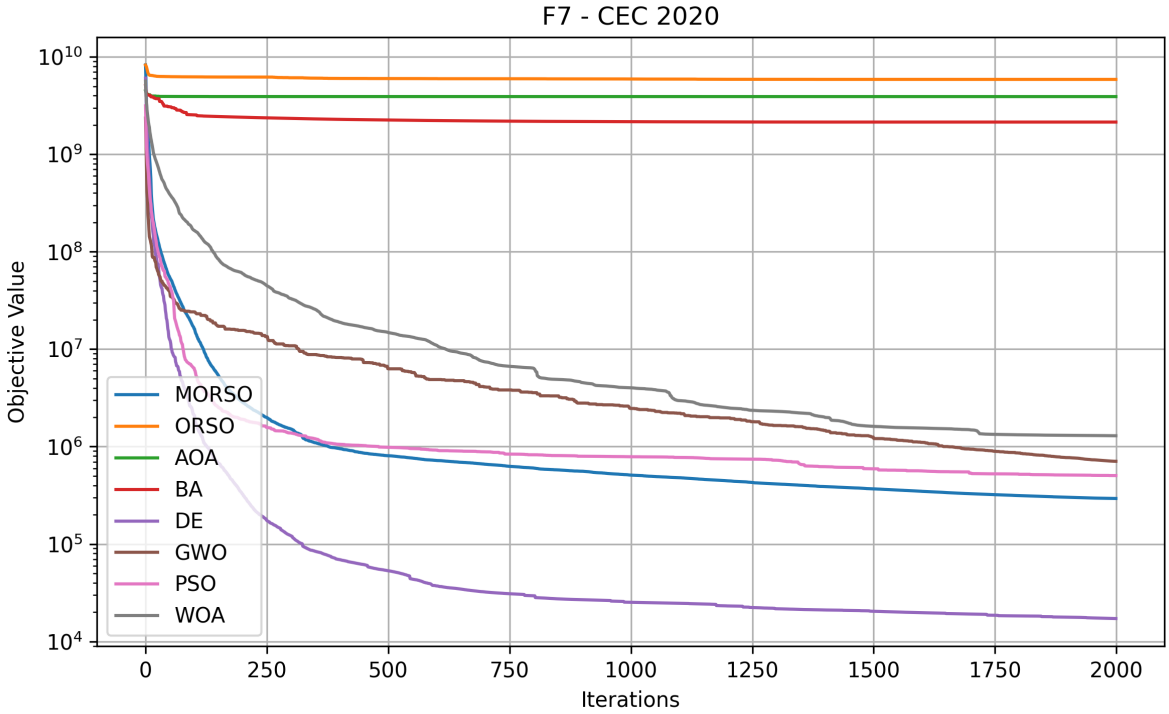


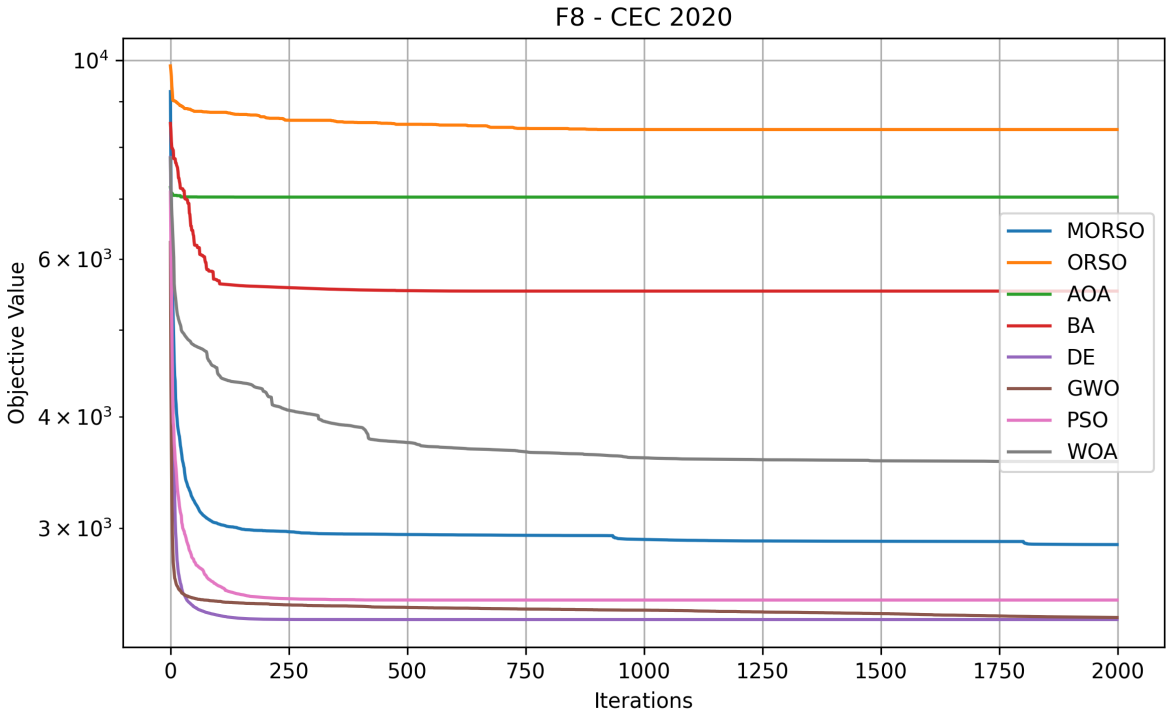

F3 - CEC 2020
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

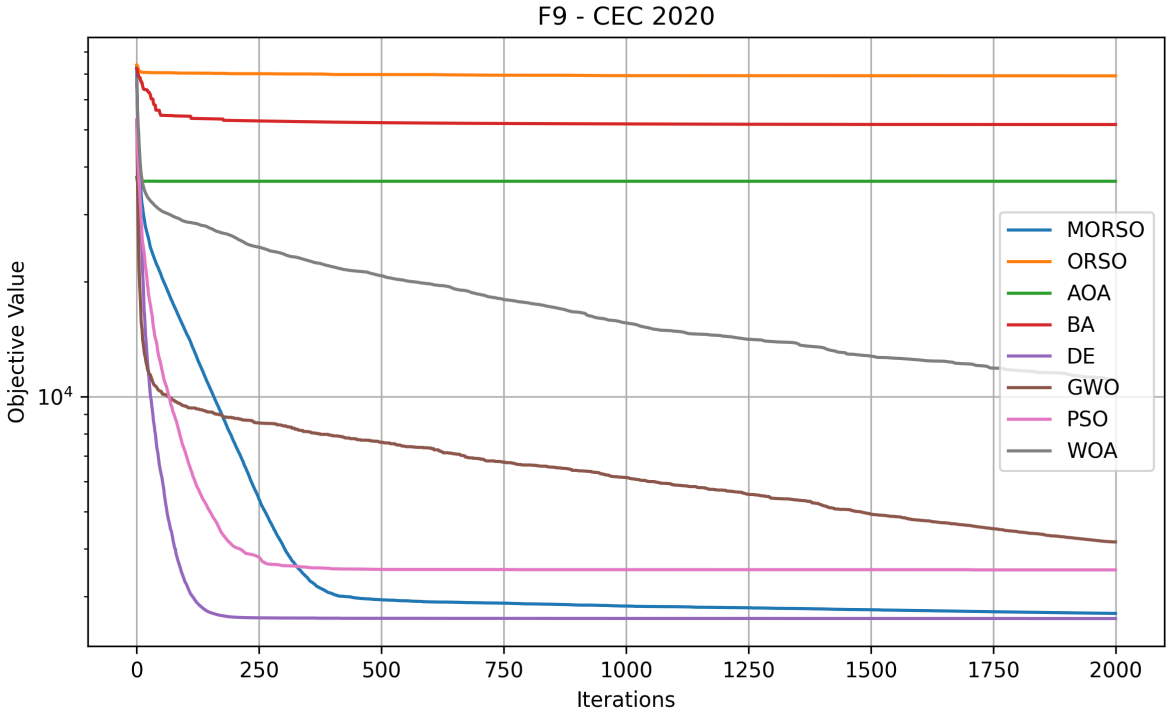

F4 - CEC 2020
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

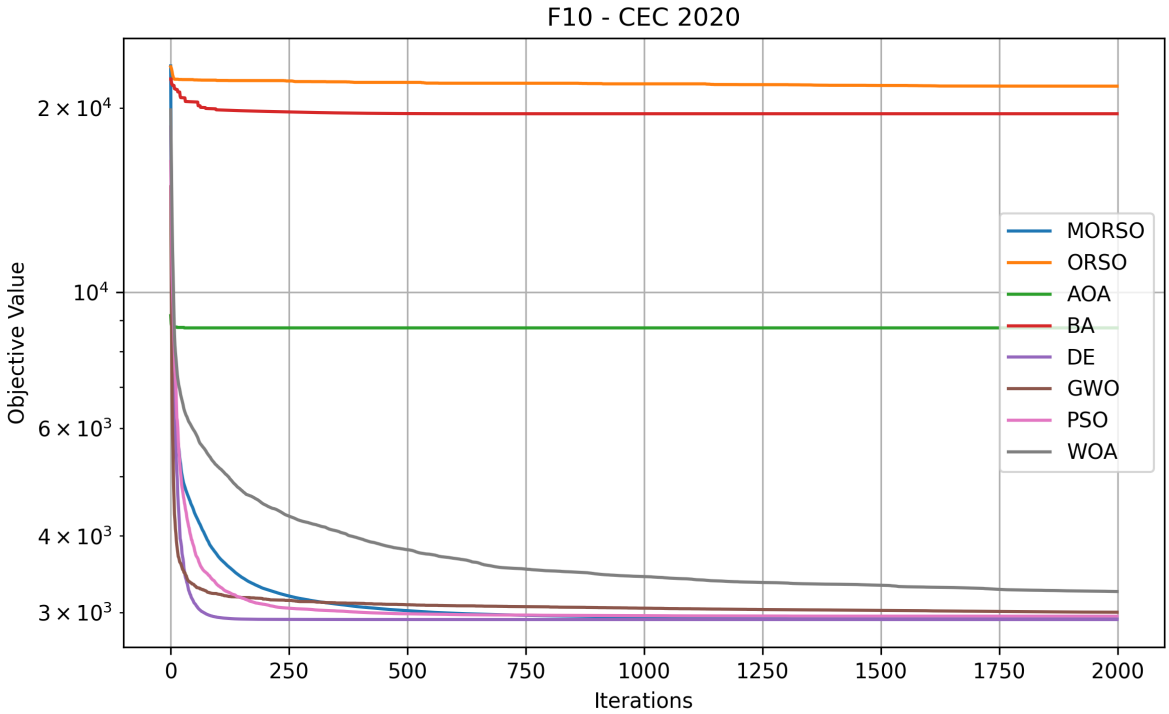

F5 - CEC 2020
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F6 - CEC 2020
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F7 - CEC 2020
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F8 - CEC 2020
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F9 - CEC 2020
Objective Value
10^4
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F10 - CEC 2020
Objective Value
2×10^4
10^4
6×10^3
4×10^3
3×10^3
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

4.6 CEC 2022 Tables

These tables give the values of the mean and standard deviation of the 12 CEC 2022 benchmark functions.

Table 11: CEC-2022 Mean Values

MOD	ORG	AOA	BA	DE	GWO	PSO	WOA
300.00	71930.54	50726.28	84236.36	300.00	477.53	300.02	2198.17
457.45	9384.68	3988.59	5711.32	436.81	474.94	467.27	539.76
600.00	602.73	602.09	602.20	600.00	600.01	600.00	600.01
970.20	1298.01	1204.52	1373.92	898.98	911.78	991.49	1134.06
903.61	931.47	914.38	931.76	900.11	901.36	905.66	906.71
6.59E+04	9.67E+09	6.81E+09	6.28E+09	23186.95	459649.15	59270.77	58129.68
2308.69	11524.34	9706.22	8569.55	2038.73	2063.19	2067.21	3507.93
2277.27	5.73E+13	2.54E+14	3.60E+13	2226.84	3489.37	3005.07	6355.93
2641.72	8943.10	9250.39	5128.97	2635.64	2670.93	2664.38	2868.39
4585.12	7405.56	7729.72	7778.46	3464.95	3423.46	3498.42	4952.57
2690.57	23201.83	14747.54	9054.31	2600.77	2609.31	2609.11	2889.62
3114.30	3894.93	4372.63	3458.28	2945.68	2961.48	3036.01	3341.09

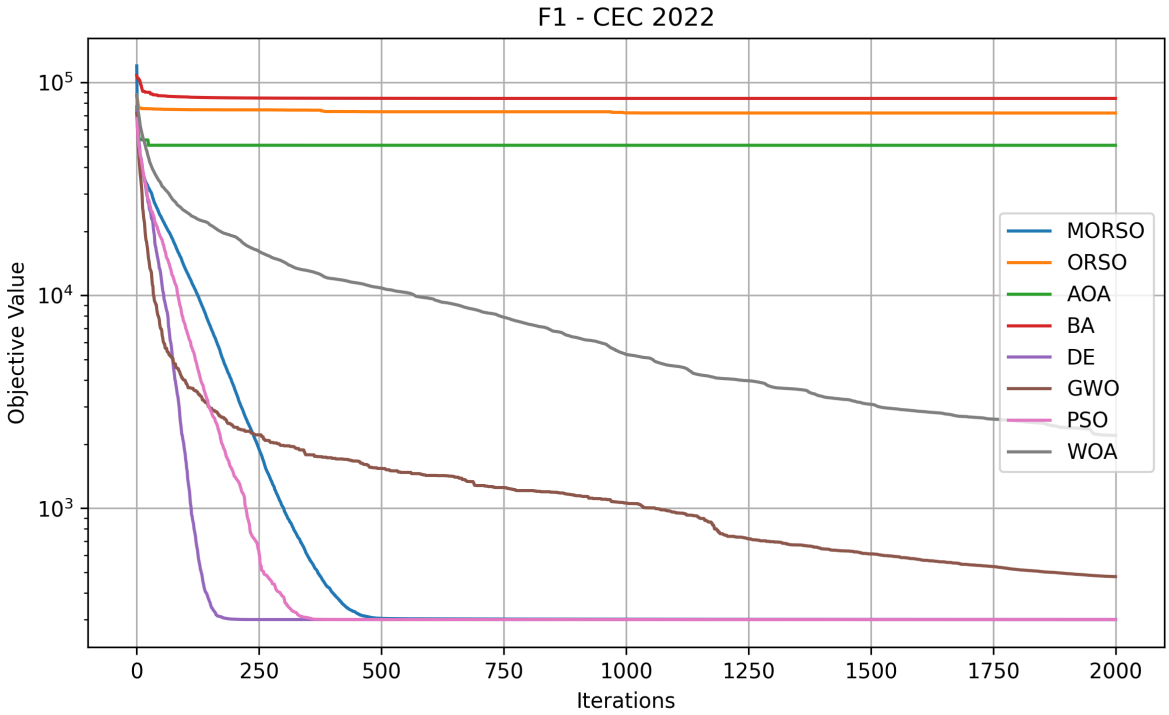
Table 12: CEC-2022 Standard Deviation Values

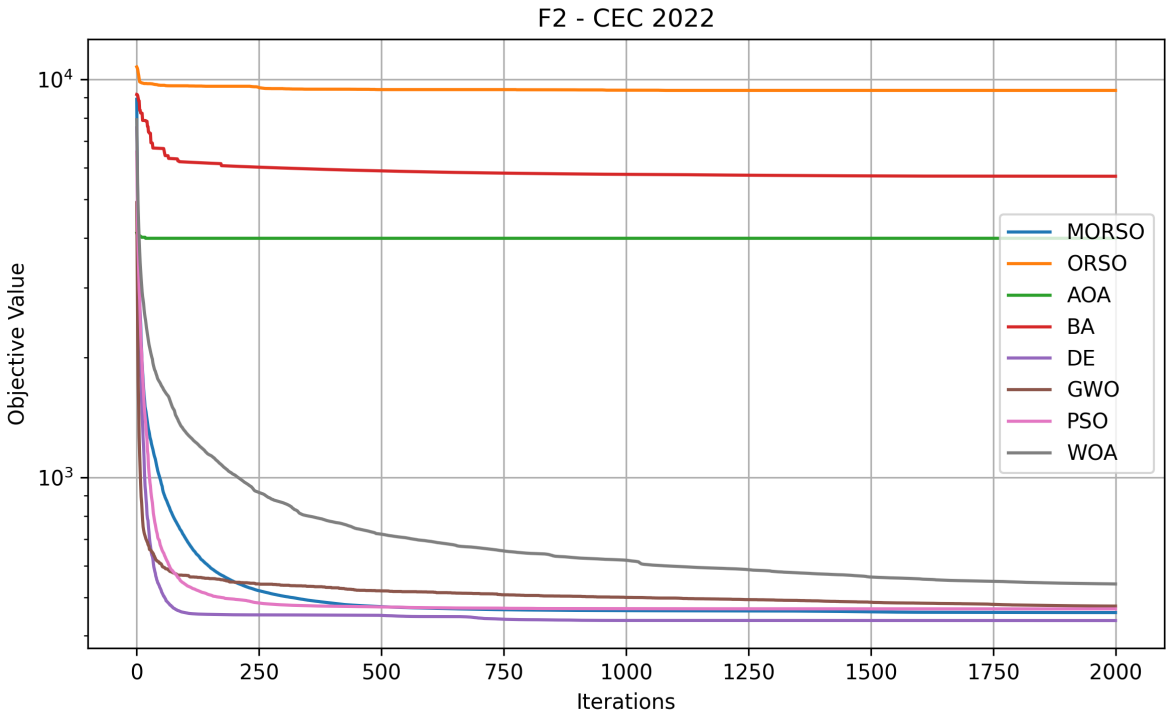
MOD	ORG	AOA	BA	DE	GWO	PSO	WOA
1.53E-04	11343.42	46794.32	21101.59	1.27E-14	256.84	0.09	1898.26
21.16	1958.79	883.90	1631.93	20.82	20.30	41.32	38.25
1.16E-08	0.40	0.18	0.54	0.00	0.01	9.52E-05	0.01
43.88	57.12	52.07	55.63	16.46	50.50	60.32	64.82
1.96	5.84	2.65	6.20	0.23	1.21	3.15	2.95
2.26E+04	4.07E+09	2.14E+09	3.58E+09	10713.43	1.74E+06	15897.34	20427.02
247.98	2252.24	2245.66	1990.69	6.47	29.53	49.18	627.49
127.09	1.75E+14	6.54E+14	1.03E+14	1.16	749.64	809.89	2179.13
5.84	1909.15	2586.38	846.99	7.61E-13	27.71	24.66	407.10
973.10	763.66	1401.21	1303.48	1148.43	958.36	896.66	1508.83
292.29	8143.46	4091.02	4097.40	3.34	7.80	8.93	661.59
115.23	400.41	286.57	191.77	10.53	20.82	46.44	199.17

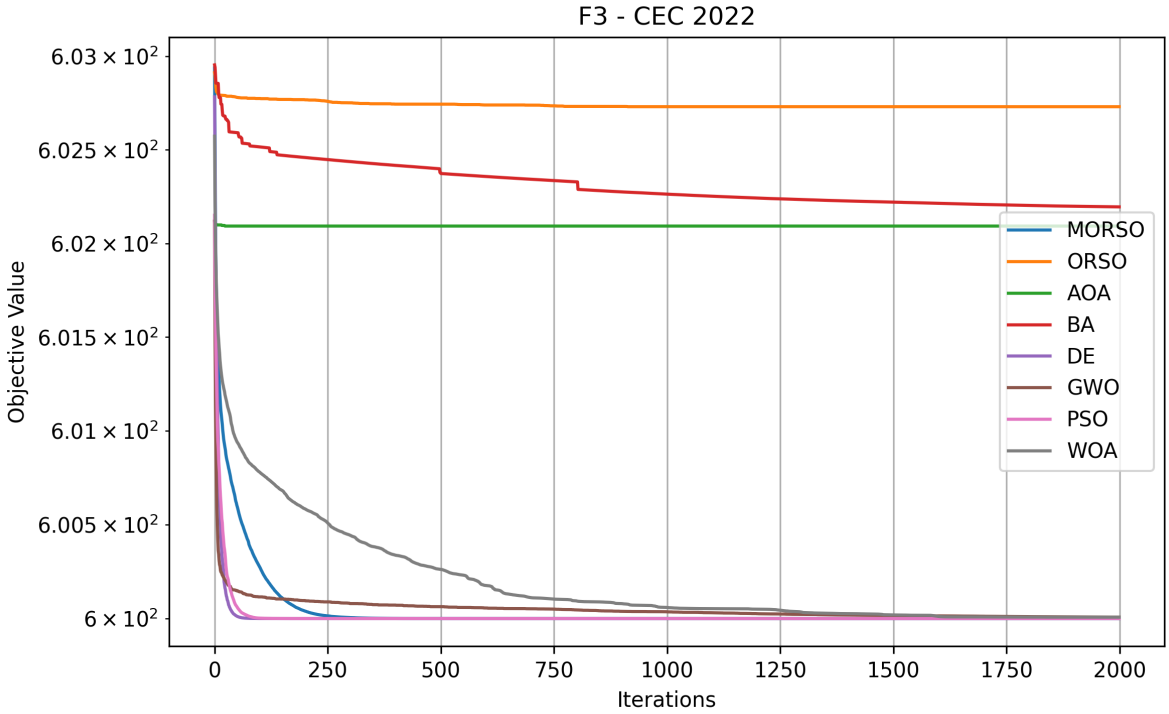
Table 13: CEC-2022 Ranks

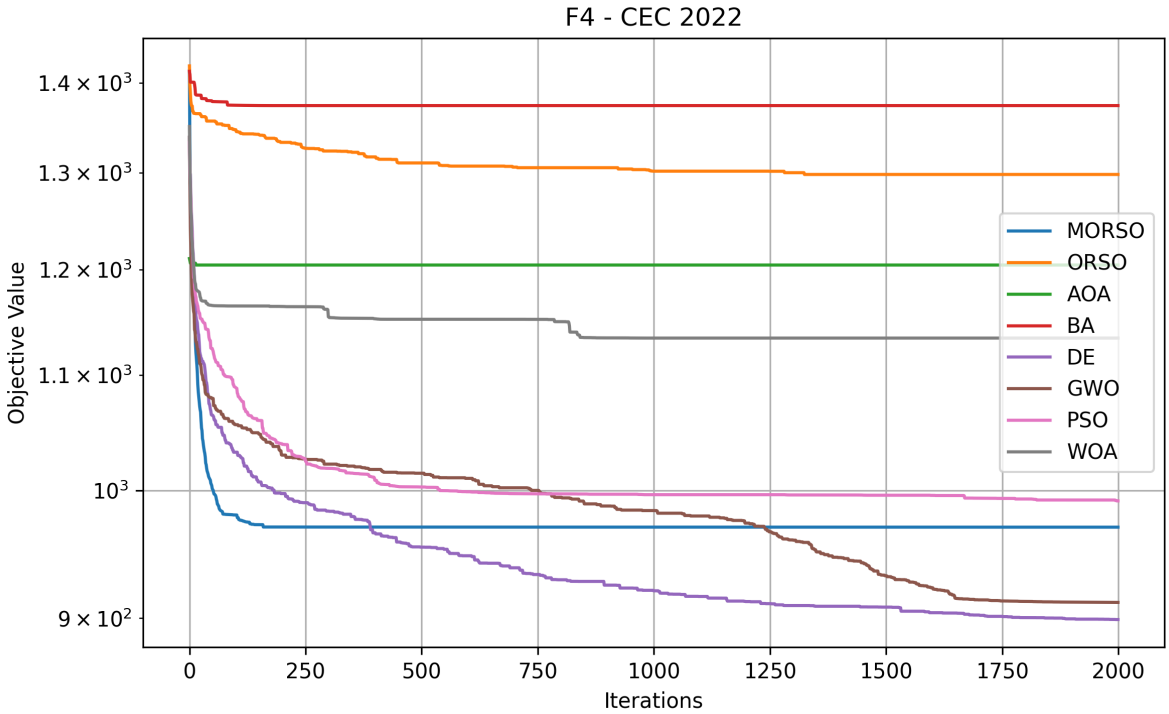
MOD	ORG	AOA	BA	DE	GWO	PSO	WOA
2	7	6	8	1	4	3	5
2	8	6	7	1	4	3	5
2	8	6	7	1	5	3	4
3	7	6	8	1	2	4	5
3	7	6	8	1	2	4	5
4	8	7	6	1	5	3	2
4	8	7	6	1	2	3	5
2	7	8	6	1	4	3	5
2	7	8	6	1	4	3	5
4	6	7	8	2	1	3	5
4	8	7	6	1	3	2	5
4	7	8	6	1	2	3	5

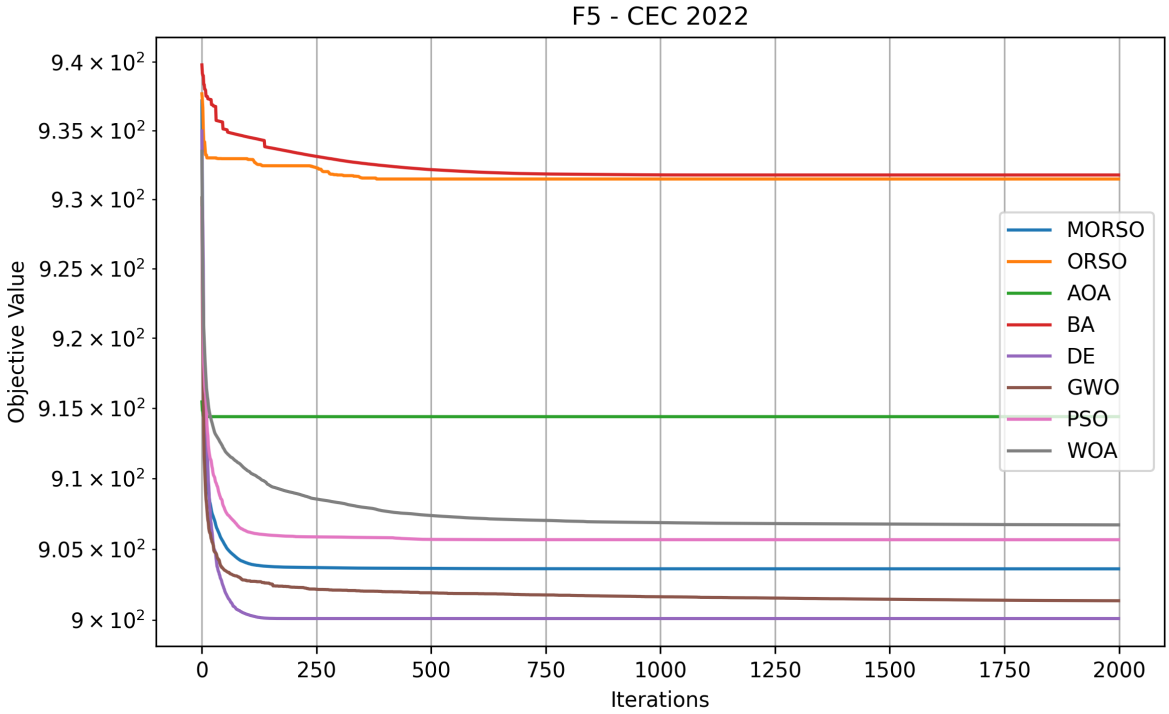
4.7 CEC 2022 Graphs

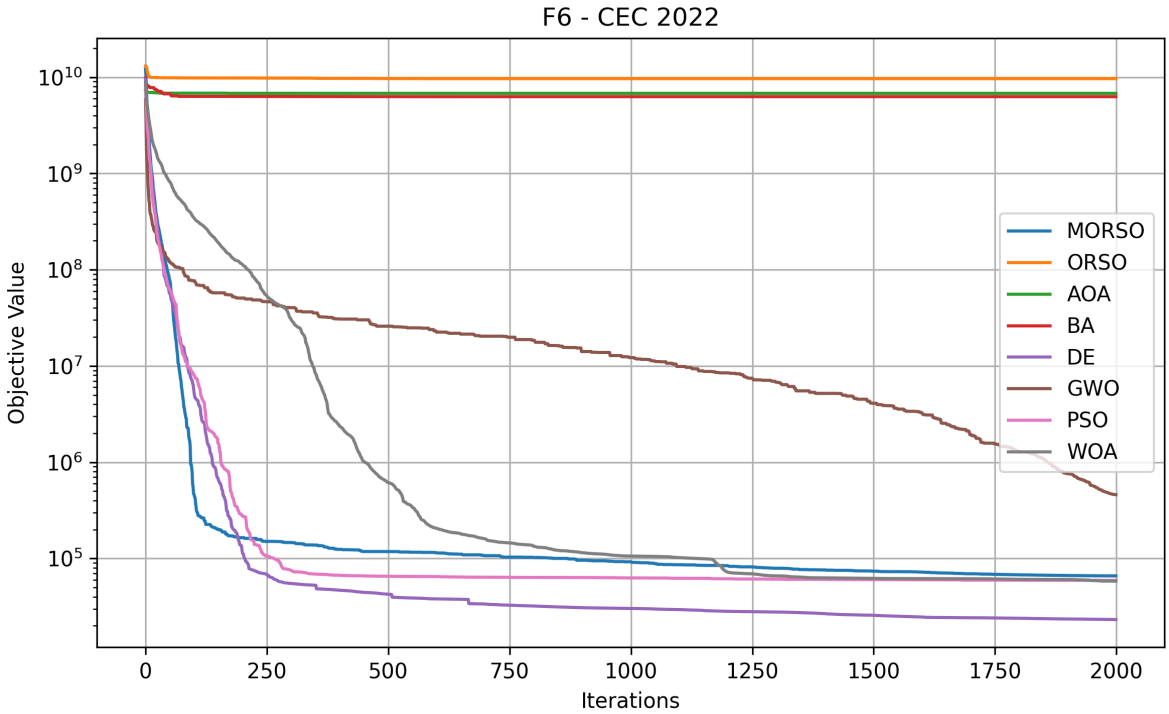


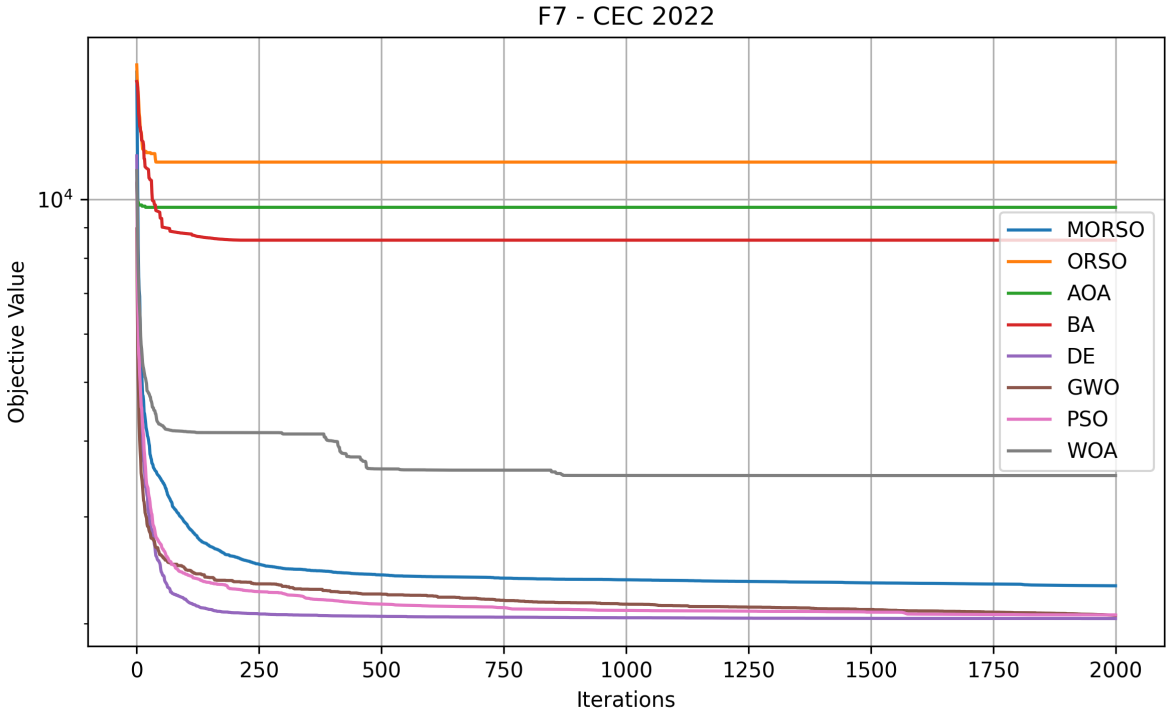


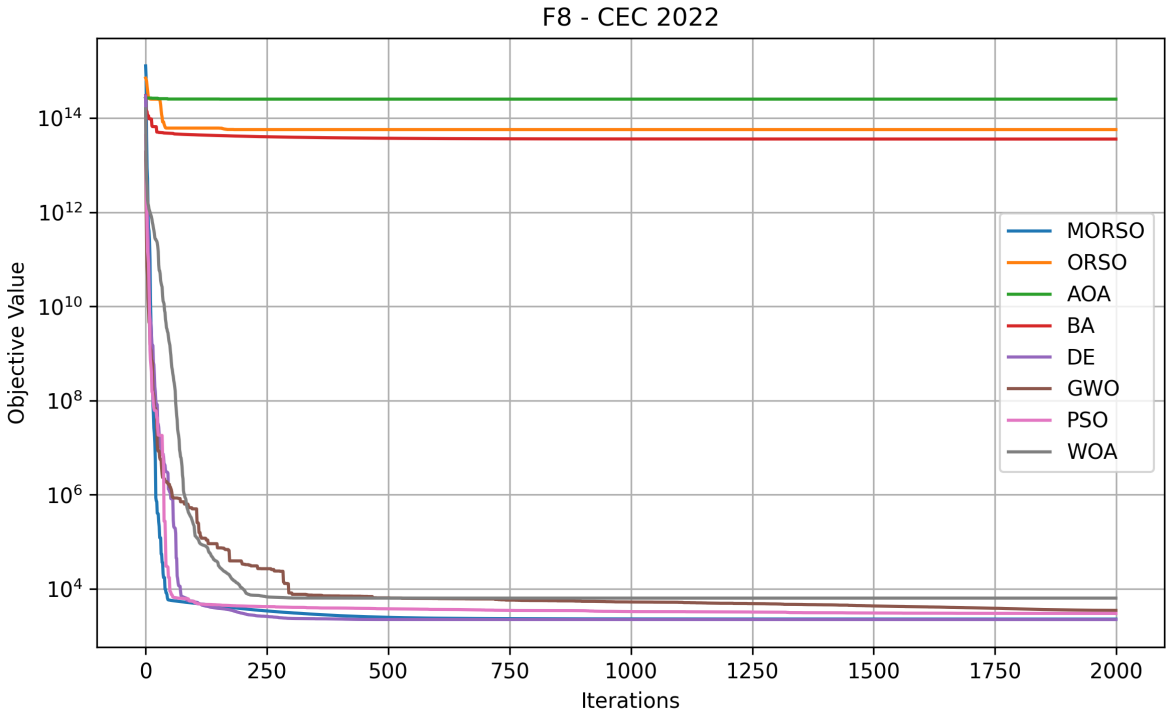

F3 - CEC 2022
Objective Value
Iterations
6.03 × 10²
6.025 × 10²
6.02 × 10²
6.015 × 10²
6.01 × 10²
6.005 × 10²
6 × 10²
0
250
500
750
1000
1250
1500
1750
2000
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

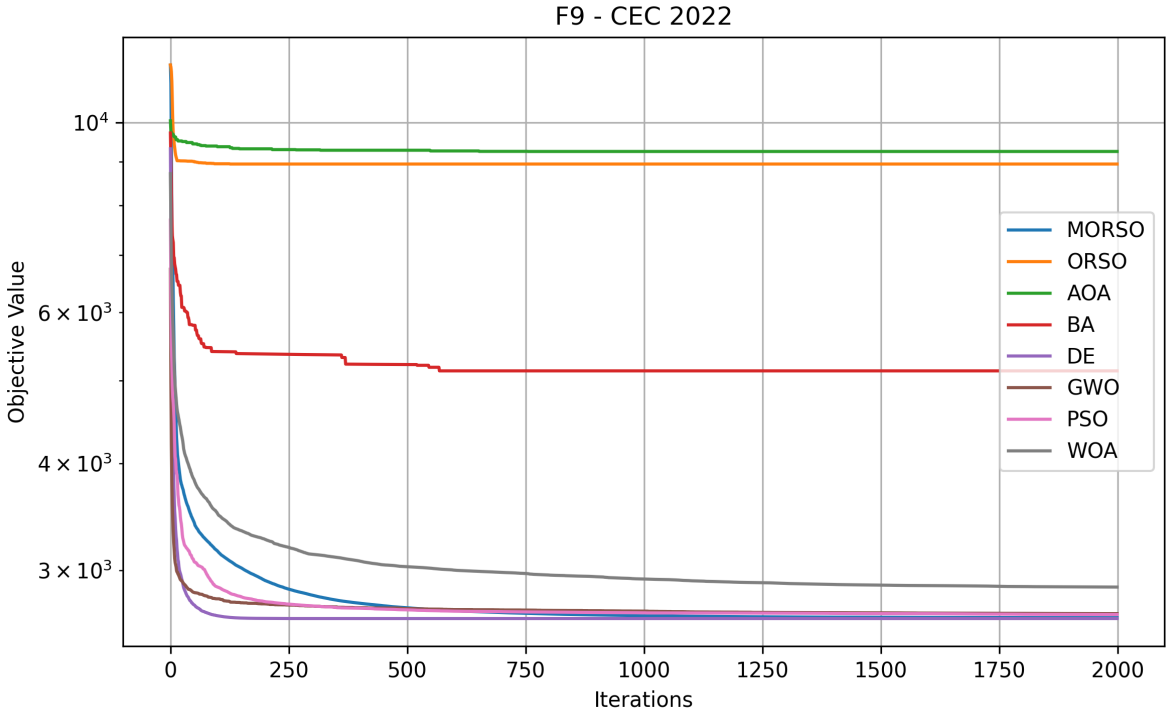

F4 - CEC 2022
Objective Value
Iterations
1.4 × 10³
1.3 × 10³
1.2 × 10³
1.1 × 10³
10³
9 × 10²
0
250
500
750
1000
1250
1500
1750
2000
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

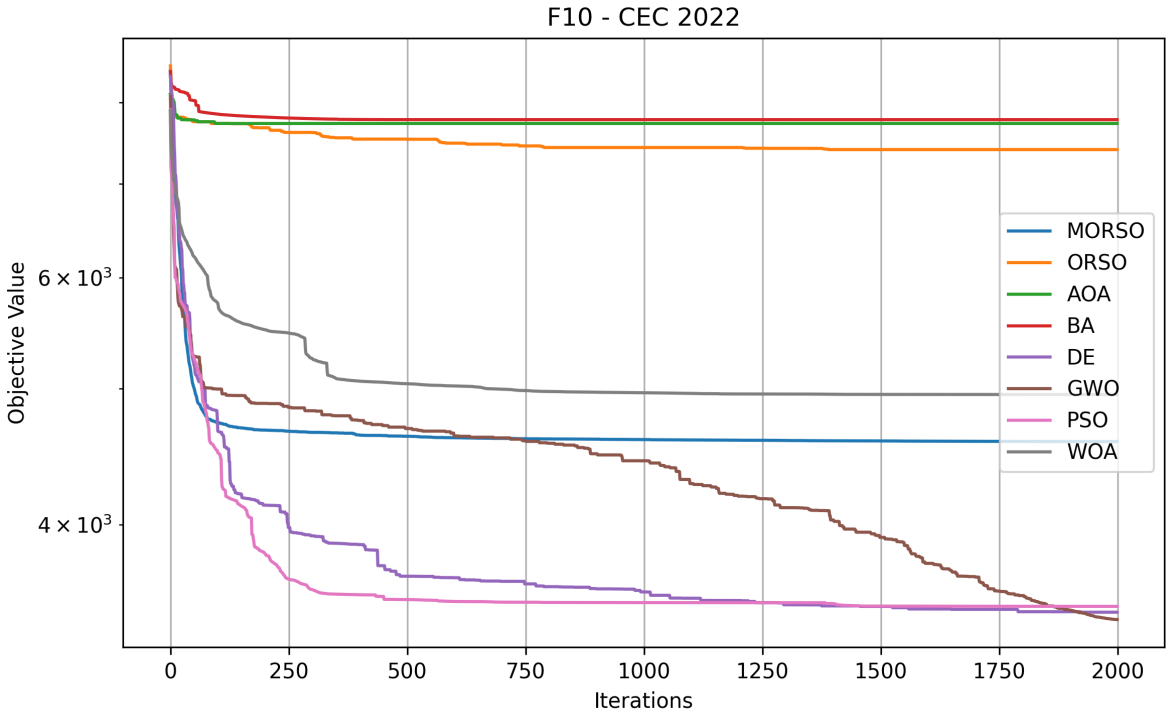

F5 - CEC 2022
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

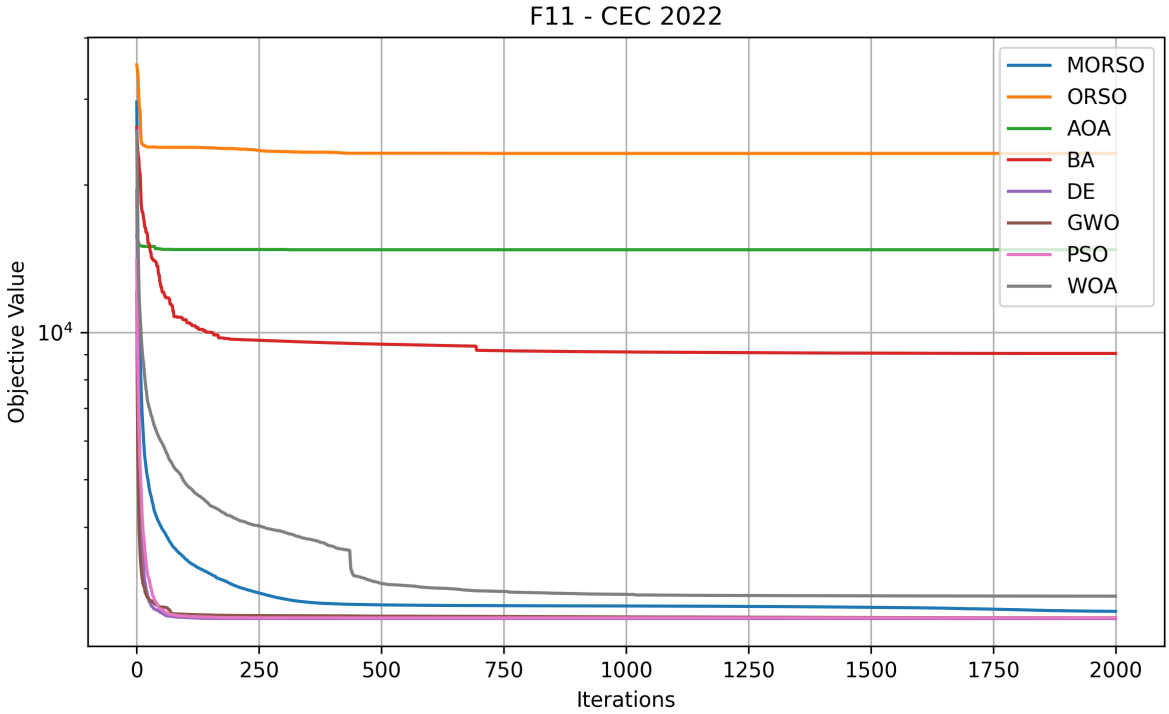

F6 - CEC 2022
Objective Value
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

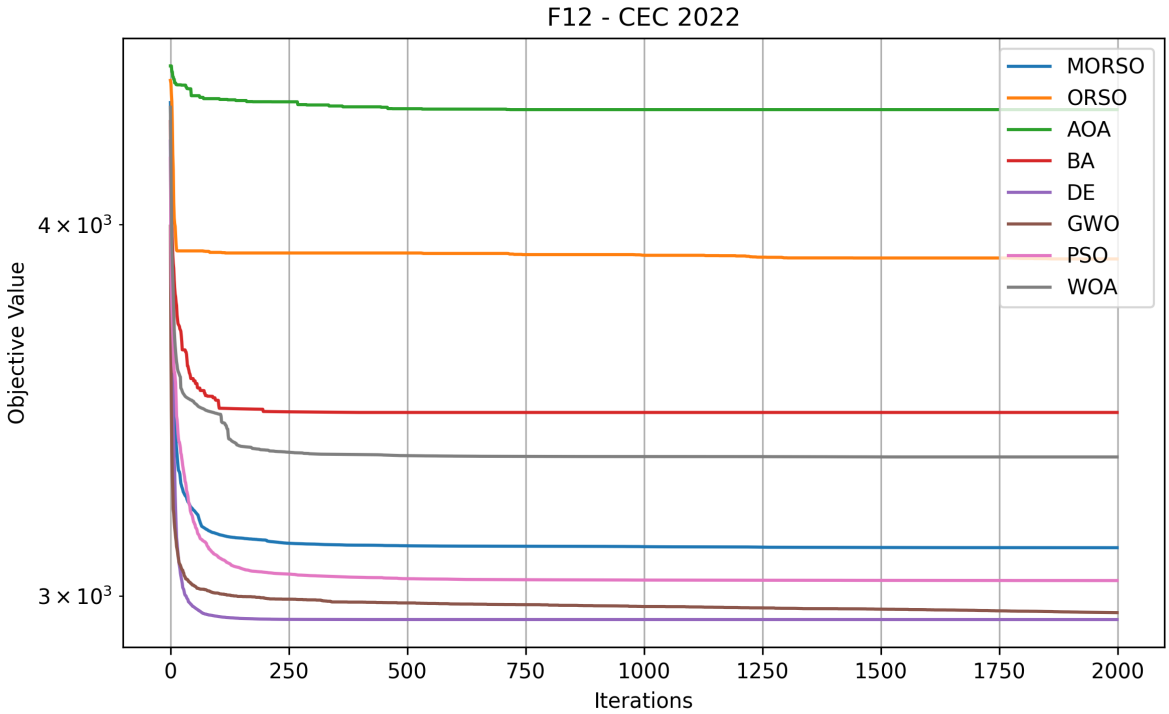

F7 - CEC 2022
Objective Value
10^4
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F8 - CEC 2022
Objective Value
10^{14}
10^{12}
10^{10}
10^8
10^6
10^4
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F9 - CEC 2022
Objective Value
10^4
6×10^3
4×10^3
3×10^3
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F10 - CEC 2022
Objective Value
6×10^3
4×10^3
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F11 - CEC 2022
Objective Value
10^4
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA


F12 - CEC 2022
Objective Value
4×10^3
3×10^3
0
250
500
750
1000
1250
1500
1750
2000
Iterations
MORSO
ORSO
AOA
BA
DE
GWO
PSO
WOA

4.8 Interpretation of the Graphs:

These graphs, plotted for Objective Value vs. Iterations, over the CEC 2020 and CEC 2022 benchmarks, give us a rich potential to study the behavior of various optimizers, with interpretations on their general performance and the time (in terms of the number of iterations) these optimizers take to reach that performance. The Modified Olive Ridley Survival Optimizer performs significantly better than the Whale Optimization Algorithm. It also performs slightly better than the Grey Wolf Optimizer. It obviously outperforms the original ORSO by a significant margin. Another caveat for it is that it generally performs slightly worse than Evolutionary Algorithms, like Differential Evolution. However, as seen clearly through the graph, DE takes much longer to train and reach the optimum value, while the MORSO is much more robust, while giving a performance only slightly worse than Differential Evolution. This shows how effective the MORSO really is.

5 Student's T-Test

5.1 Student's T-Test(Modified with respect to Original): CEC 2014 Functions

Table 14: Student's T-Test Results for CEC 2014 Functions

CEC_2014_Function	P-Value	Improvement(Yes/No)
CEC_2014_Function_1	7.68533845801398E-27	Yes
CEC_2014_Function_2	7.99731061548101E-44	Yes
CEC_2014_Function_3	0.000796310258957583	Yes
CEC_2014_Function_4	4.12443341727609E-29	Yes
CEC_2014_Function_5	1.45876499463638E-59	Yes
CEC_2014_Function_6	6.24082981708231E-38	Yes
CEC_2014_Function_7	4.7508347735741E-41	Yes
CEC_2014_Function_8	8.65476570520341E-37	Yes
CEC_2014_Function_9	8.32730339735814E-40	Yes
CEC_2014_Function_10	7.71359918680843E-37	Yes
CEC_2014_Function_11	4.39189305553016E-38	Yes
CEC_2014_Function_12	5.94491195700781E-44	Yes
CEC_2014_Function_13	3.43232304836803E-45	Yes
CEC_2014_Function_14	4.01823313187723E-42	Yes
CEC_2014_Function_15	3.3149547805112E-19	Yes
CEC_2014_Function_16	1.91333948599718E-14	Yes
CEC_2014_Function_17	3.38645345533017E-19	Yes
CEC_2014_Function_18	6.3882556940308E-26	Yes
CEC_2014_Function_19	3.08880611859735E-13	Yes
CEC_2014_Function_20	0.0000194582589575555	Yes
CEC_2014_Function_21	1.13408587058262E-15	Yes
CEC_2014_Function_22	0.00000178110403903572	Yes
CEC_2014_Function_23	1.7052738406974E-21	Yes
CEC_2014_Function_24	3.35257970878298E-45	Yes
CEC_2014_Function_25	5.68223544063738E-26	Yes
CEC_2014_Function_26	0.0000000413838946992443	Yes
CEC_2014_Function_27	1.58289975792558E-27	Yes
CEC_2014_Function_28	1.80776496759117E-22	Yes
CEC_2014_Function_29	7.83515827465297E-20	Yes
CEC_2014_Function_30	0.000341348416483089	Yes

5.2 Student's T-Test(Modified with respect to Original): CEC 2017 Functions

Table 15: Student's T-Test Results for CEC 2017 Functions

CEC_2017_Function	P-Value	Improvement(Yes/No)
CEC_2017_Function_1	0.00651285867674062	Yes
CEC_2017_Function_2	NA	NA
CEC_2017_Function_3	1.20287074590391E-15	Yes
CEC_2017_Function_4	0.000000567553082806162	Yes
CEC_2017_Function_5	1.0017969368758E-34	Yes
CEC_2017_Function_6	2.83376224031942E-35	Yes
CEC_2017_Function_7	7.25615409642761E-39	Yes
CEC_2017_Function_8	1.04070900072919E-43	Yes
CEC_2017_Function_9	7.31049444113517E-36	Yes
CEC_2017_Function_10	1.18031900172795E-14	Yes
CEC_2017_Function_11	3.4562437800668E-28	Yes
CEC_2017_Function_12	9.94929700031837E-33	Yes
CEC_2017_Function_13	9.23392158651038E-15	Yes
CEC_2017_Function_14	2.10232824665059E-28	Yes
CEC_2017_Function_15	2.33882395378284E-17	Yes
CEC_2017_Function_16	0.00651285867672536	Yes
CEC_2017_Function_17	1.26903874342512E-15	Yes
CEC_2017_Function_18	0.000000567553082875738	Yes
CEC_2017_Function_19	1.12117878063909E-31	Yes
CEC_2017_Function_20	6.44754036847912E-35	Yes
CEC_2017_Function_21	5.36082587088936E-26	Yes
CEC_2017_Function_22	2.21819377759931E-43	Yes
CEC_2017_Function_23	4.82653789945297E-46	Yes
CEC_2017_Function_24	3.07296794506057E-27	Yes
CEC_2017_Function_25	2.48646709191568E-16	Yes
CEC_2017_Function_26	2.79220027933219E-22	Yes
CEC_2017_Function_27	7.48850738172642E-29	Yes
CEC_2017_Function_28	0.00000311298392053561	Yes
CEC_2017_Function_29	0.000000125412694397631	Yes
CEC_2017_Function_30	NA	NA

5.3 Student's T-Test(Modified with respect to Original): CEC 2020 Functions

Table 16: Student's T-Test Results for CEC 2020 Functions

CEC_2020_Function	P-Value	Improvement(Yes/No)
CEC_2020_Function_1	1.35189361030933E-16	Yes
CEC_2020_Function_2	2.61937452358862E-14	Yes
CEC_2020_Function_3	1.13493544374922E-20	Yes
CEC_2020_Function_4	0.0000000699328324446071	Yes
CEC_2020_Function_5	0.000000313764546067343	Yes
CEC_2020_Function_6	0.0000000913516233372607	Yes
CEC_2020_Function_7	0.0000025240535789843	Yes
CEC_2020_Function_8	2.38056598958581E-13	Yes
CEC_2020_Function_9	1.44079217530675E-16	Yes
CEC_2020_Function_10	1.29222298358088E-11	Yes

5.4 Student's T-Test(Modified with respect to Original): CEC 2022 Functions

Table 17: Student's T-Test Results for CEC 2022 Functions

CEC_2022_Function	P-Value	Improvement(Yes/No)
CEC_2022_Function_1	8.97007336983702E-17	Yes
CEC_2022_Function_2	3.4878320608131E-14	Yes
CEC_2022_Function_3	2.22919469481455E-17	Yes
CEC_2022_Function_4	8.56025411413133E-15	Yes
CEC_2022_Function_5	6.00602962026339E-14	Yes
CEC_2022_Function_6	0.00000000296339459164891	Yes
CEC_2022_Function_7	2.15448110993095E-13	Yes
CEC_2022_Function_8	0.169168971163424	Yes
CEC_2022_Function_9	1.13734070454407E-11	Yes
CEC_2022_Function_10	0.000000174033094032764	Yes
CEC_2022_Function_11	0.0000000109836795812079	Yes
CEC_2022_Function_12	0.00000193598874891523	Yes

6 Wilcoxon Test for Significance Analysis

6.1 Wilcoxon Test for Significance Analysis(Modified with respect to Original): CEC 2014 Functions

Table 18: Wilcoxon Test Results for CEC 2014 Functions

CEC_2014_Function	P-Value	Improvement(Yes/No)
CEC_2014_Function_1	7.555221E-10	Yes
CEC_2014_Function_2	1.77635683940025E-15	Yes
CEC_2014_Function_3	1.776357E-15	Yes
CEC_2014_Function_4	1.77635683940025E-15	Yes
CEC_2014_Function_5	1.77635683940025E-15	Yes
CEC_2014_Function_6	1.77635683940025E-15	Yes
CEC_2014_Function_7	1.77635683940025E-15	Yes
CEC_2014_Function_8	0.000000000755522087638067	Yes
CEC_2014_Function_9	1.77635683940025E-15	Yes
CEC_2014_Function_10	0.0000000007553512643873 76	Yes
CEC_2014_Function_11	1.77635683940025E-15	Yes
CEC_2014_Function_12	1.77635683940025E-15	Yes
CEC_2014_Function_13	1.77635683940025E-15	Yes
CEC_2014_Function_14	1.77635683940025E-15	Yes
CEC_2014_Function_15	1.77635683940025E-15	Yes
CEC_2014_Function_16	8.88178419700125E-15	Yes
CEC_2014_Function_17	1.77635683940025E-15	Yes
CEC_2014_Function_18	1.77635683940025E-15	Yes
CEC_2014_Function_19	1.77635683940025E-15	Yes
CEC_2014_Function_20	1.77635683940025E-15	Yes
CEC_2014_Function_21	1.77635683940025E-15	Yes
CEC_2014_Function_22	1.77635683940025E-15	Yes
CEC_2014_Function_23	1.77635683940025E-15	Yes
CEC_2014_Function_24	0.0000000007553512643873 76	Yes
CEC_2014_Function_25	1.77635683940025E-15	Yes
CEC_2014_Function_26	0.000000001756896850224 62	Yes
CEC_2014_Function_27	1.77635683940025E-15	Yes
CEC_2014_Function_28	0.000000000802928068023497	Yes
CEC_2014_Function_29	0.0000000007553512643873 76	Yes
CEC_2014_Function_30	1.77635683940025e-15	Yes

6.2 Wilcoxon Test for Significance Analysis(Modified with respect to Original): CEC 2017 Functions

Table 19: Wilcoxon Test Results for CEC 2017 Functions

CEC_2017_Function	P-Value	Improvement(Yes/No)
CEC_2017_Function_1	1.77635683940025E-15	Yes
CEC_2017_Function_2	NA	NA
CEC_2017_Function_3	0.00000000075552208763806 7	Yes
CEC_2017_Function_4	1.77635683940025E-15	Yes
CEC_2017_Function_5	1.77635683940025E-15	Yes
CEC_2017_Function_6	0.00000000075552208763806 7	Yes
CEC_2017_Function_7	0.00000000075552208763806 7	Yes
CEC_2017_Function_8	1.77635683940025E-15	Yes
CEC_2017_Function_9	1.77635683940025E-15	Yes
CEC_2017_Function_10	0.00000000075535126438737 6	Yes
CEC_2017_Function_11	1.77635683940025E-15	Yes
CEC_2017_Function_12	1.77635683940025E-15	Yes
CEC_2017_Function_13	0.00000000075552208763806 7	Yes
CEC_2017_Function_14	0.00000000075535126438737 6	Yes
CEC_2017_Function_15	1.77635683940025E-15	Yes
CEC_2017_Function_16	0.00000000075552208763806 7	Yes
CEC_2017_Function_17	1.77635683940025E-15	Yes
CEC_2017_Function_18	1.77635683940025E-15	Yes
CEC_2017_Function_19	1.77635683940025E-15	Yes
CEC_2017_Function_20	1.77635683940025E-15	Yes
CEC_2017_Function_21	0.00000000096301008016436	Yes
CEC_2017_Function_22	1.77635683940025E-15	Yes
CEC_2017_Function_23	1.77635683940025E-15	Yes
CEC_2017_Function_24	0.00000000075552208763806 7	Yes
CEC_2017_Function_25	1.77635683940025E-15	Yes
CEC_2017_Function_26	1.77635683940025E-15	Yes
CEC_2017_Function_27	0.00000000075535126438737 6	Yes
CEC_2017_Function_28	1.77635683940025E-15	Yes
CEC_2017_Function_29	1.77635683940025E-15	Yes
CEC_2017_Function_30	NA	NA

6.3 Wilcoxon Test for Significance Analysis(Modified with respect to Original): CEC 2020 Functions

Table 20: Wilcoxon Test Results for CEC 2020 Functions

CEC_2020_Function	P-Value	Improvement(Yes/No)
CEC_2020_Function_1	0.0000019073486328125	Yes
CEC_2020_Function_2	0.0000019073486328125	Yes
CEC_2020_Function_3	0.0000019073486328125	Yes
CEC_2020_Function_4	0.0000019073486328125	Yes
CEC_2020_Function_5	0.0000019073486328125	Yes
CEC_2020_Function_6	0.0000019073486328125	Yes
CEC_2020_Function_7	0.0000019073486328125	Yes
CEC_2020_Function_8	0.0000019073486328125	Yes
CEC_2020_Function_9	0.0000019073486328125	Yes
CEC_2020_Function_10	0.0000019073486328125	Yes

6.4 Wilcoxon Test for Significance Analysis(Modified with respect to Original): CEC 2022 Functions

Table 21: Wilcoxon Test Results for CEC 2022 Functions

CEC_2022_Function	P-Value	Improvement(Yes/No)
CEC_2022_Function_1	0.0000019073486328125	Yes
CEC_2022_Function_2	0.0000019073486328125	Yes
CEC_2022_Function_3	0.0000019073486328125	Yes
CEC_2022_Function_4	0.0000019073486328125	Yes
CEC_2022_Function_5	0.0000019073486328125	Yes
CEC_2022_Function_6	0.0000019073486328125	Yes
CEC_2022_Function_7	0.0000019073486328125	Yes
CEC_2022_Function_8	0.0000019073486328125	Yes
CEC_2022_Function_9	0.0000019073486328125	Yes
CEC_2022_Function_10	0.000003814697265625	Yes
CEC_2022_Function_11	0.0000019073486328125	Yes
CEC_2022_Function_12	0.0000019073486328125	Yes

7 Examples and Engineering Problems

7.1 Comparing various NIAs on the basis of 3 Engineering Problems: PVD(Pressure Vessel Design), WBD(Welded Beam Design), and SD(Spring Design)

Table 22: Parameters and their range constraints for engineering problems and obtained optimum values

EP	Parameters_Range_Optimum_Value	ORS	MORS	TSA	MVO	SCA	GWO	WOA	BA	DE
PVD	T_s(0,99)	0.4247	0.9127	0.6347	0.4311	0.6347	0.4347	0.4347	0.9347	0.5347
PVD	T_h(0,99)	0.8031	0.4316	0.8471	0.8201	1.234	0.8231	0.8231	1.6231	0.8431
PVD	R(10,200)	42.2675	49.9853	46.2875	42.2756	47.2843	42.2875	42.2875	84.2875	44.2875
PVD	L(10,200)	176.3267	98.4616	179.3789	176.1267	181.2587	176.3567	176.3567	192.3567	179.3567
PVD	Optimum_Value	6031.927	6036.5071	6462.522	7007.709	7884.301	6052.345	7011.218	17595.43	6302.13
WBD	h(0,1,2)	0.106432	0.12781616	0.109531	0.1094531	0.109434	0.20435	0.20531	0.4643	0.10531
WBD	l(0,1,10)	2.9118	3.57671445	2.942	3.1142	3.1102	3.4641	3.4742	5.4342	3.1742
WBD	t(0,1,10)	8.0976	9.99952275	8.1763	8.9764	8.8763	9.0264	9.0364	12.9876	8.9364
WBD	b(0,1,2)	0.10761	0.16807178	0.10671	0.10962	0.10861	0.2058	0.2061	0.4562	0.1061
WBD	Optimum_Value	1.476776	1.48264868977387	1.507568	1.62	1.616402	1.7249	1.74	3.3	1.636818
SD	d(0.05,2)	0.05062	0.05235807	0.05524	0.06521	0.05223	0.05169	0.05424	0.1721	0.1521
SD	D(0.25,1.3)	0.35317	0.37858038	0.366838	0.386234	0.366231	0.356737	0.366738	1.26235	0.96234
SD	N(2.15)	11.25882	11.36972252	11.2983	11.4383	11.2943	11.28885	11.2967	13.4139	12.4231
SD	Optimum_Value	0.010931	0.0129268053213524	0.012798	0.013428	0.012438	0.011187	0.012256	0.033267	0.021145

Interpretation of Table Values: The Modified ORSO algorithm demonstrates optimal performance, closely matching the efficiency and effectiveness of the original ORSO and the seven other optimization algorithms discussed in the base paper. While the final objective values achieved are comparable across all methods, the Modified ORSO consistently converges to different sets of parameter values. Interestingly, despite these variations in the solution vectors, all parameter values generated by the Modified ORSO remain well within the defined constraints of the respective optimization problems. This behavior indicates that the algorithm explores alternative feasible regions in the search space while still maintaining optimality. Such diversity in solution pathways can be advantageous in multi-modal landscapes or in real-world applications where alternate optimal configurations may offer practical benefits. Furthermore, this variation highlights the robustness and exploratory strength of the Modified ORSO in identifying multiple viable optima. The algorithm avoids premature convergence and maintains a balance between exploration and exploitation. As a result, it offers a competitive alternative to traditional optimization methods. Overall, the Modified ORSO proves to be both effective and reliable across different problem scenarios.

7.2 Final Example: Basic Quadratic Function nearly optimized in 10 iterations

```
#Example:- Basic quadratic function nearly optimized in 10
   iterations
def objective_function(x):
  return x[0]**2 + 3 * x[0] + 2
morso = ModifiedOliveRidleySurvivalOptimizer(
    objective_function=objective_function,
    num_turtles=30,
    dimensions=1,
    lower_bound=-10,
    upper_bound=10,
    max_iterations=10
)
best_x, best_fitness,convergence = morso.optimize()
print(f"Minimum value of the function is {best_fitness} at x = {
   best_x[0]}")
```

```
Minimum value of the function is -0.2496681525277813 at x = -1.4817833188473268
```