Universiteit Leiden
The Netherlands

Computer Science

Illuminating the EC-Bestiary:
Conceptual Analysis and Benchmarking of the
Invasive Weed Optimization Algorithm

Barmad Shreef

Supervisors:
Dr. Elena Raponi & Haoran Yin

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/06/2024

Abstract

This thesis delves into a novel numerical stochastic optimization algorithm inspired by colonizing weeds. These plants are known for their vigorous, invasive habits of growth, which pose a serious threat to desirable plants, making them a threat to agriculture. They were found to be highly adaptable to changes in the environment. It is tried to mimic these characteristics in a simple but effective optimizing algorithm named Invasive Weed Optimization (IWO).
In this thesis, the feasibility, efficiency, and effectiveness of IWO are tested in detail through a set of benchmark multi-dimensional test functions facilitated by the IOHprofiler, a comprehensive benchmarking platform designed to evaluate the performance of iterative optimization heuristics (IOHs).
We investigate the best parameter combinations for enhancing the algorithm's adaptability and performance across various scenarios. In addition, we provide a comprehensive comparison with other recent evolutionary-based algorithms, including genetic algorithms, augmented covariance matrix adaptation evolution strategy and bimodal population covariance matrix adaptation evolution strategy.
The IWO performs well across a range of test functions, demonstrating competitive convergence rates and robustness, particularly in lower-dimensional problems. While IWO excels in its exploitation phase, making it effective in finding optimal or near-optimal solutions, its performance tends to decline in higher-dimensional spaces compared to other algorithms.

Contents

1 Introduction

1.1 Background

Many major problems involve determining the most efficient way to do a task. This frequently entails determining the maximum or minimum value of a function, such as the minimum time required to complete a particular journey, the minimum cost of doing a task, the maximum power that a device can produce, and so on. Most of these problems can be dealt with by determining a proper function and then applying mathematical techniques to determine the maximum or minimum value required. However, when we have limited knowledge about the function we are attempting to optimize, we use meta-heuristic search algorithms[Sör15], which are algorithms inspired by natural systems, to solve complex optimization problems.
The use of algorithms inspired by natural processes and/or occurrences to solve optimization problems has received a lot of attention. For example, Genetic Algorithms (GAs)[Hol75], are standard optimization tools. Other numerical direct search optimization approaches include Simulated Annealing (SA)[KGV83], Ant Colony Optimization (ACO)[DMC96], and Particle Swarm Optimization (PSO)[EK95].
The EC-Bestiary encompasses a diverse collection of bio-inspired optimization techniques that draw their principles from natural phenomena [CA21]. Over time, it has expanded to include numerous newly developed evolutionary algorithms, each inspired by different aspects of nature.

1.2 Objectives

This thesis aims to delve into the mechanisms of one specific EC-Bestiary algorithm, namely Invasive Weed Optimization Algorithm (IWO), proposed by Mehrabian and Lucas[ML06], inspired by an invasive weed colonization phenomena that is frequently observed in agriculture. The authors claim that despite its simplicity, the algorithm has demonstrated its effectiveness in reaching the optimal solution by utilizing fundamental characteristics of a weed colony, such as seeding, growth, and competition.
The primary objective of this thesis is an in-depth examination of its components and underlying principles. The research will involve the practical implementation of the algorithm, followed by a benchmarking study to evaluate its performance utilizing the IOHprofiler[DWY+18]. Through this analysis, this thesis aims to contribute to the existing body of knowledge on nature-inspired optimization algorithms.

1.3 Thesis overview

This thesis provides a comprehensive exploration of the IWO algorithm, and is structured as follows. Section 1 introduces the study's context and aims. Section 2 reviews existing literature and other relevant information. Section 3 details the IWO algorithm's mechanics and implementation. Section 4 describes the experimental setup and analyses the obtained results. Section 5 summarizes the key findings and suggests directions for future research.
This bachelor thesis is supervised by Dr. Elena Raponi at the Leiden Institute of Advanced Computer Science (LIACS).

2 Related Work

Optimization algorithms are essential tools in solving a wide range of complex problems across various domains, such as engineering, economics, and scientific research. Nature-inspired algorithms, in particular, have gained significant attention due to their ability to effectively explore large, multi-dimensional search spaces and find near-optimal solutions. These algorithms are modeled after natural processes and behaviors, providing robust mechanisms for optimization tasks.

2.1 Invasive Weed Optimization

The IWO algorithm was introduced by Mehrabian and Lucas in 2006 [ML06]. It is inspired by the colonizing behavior of weeds, which are known for their ability to adapt and spread rapidly in diverse environments. IWO mimics these characteristics by employing mechanisms such as seeding, growth, and competition to explore the search space and converge on optimal solutions. In the next chapter, we will provide a comprehensive description of the algorithm.

2.2 Compared nature-inspired algorithms

Benchmarking is crucial for evaluating the performance of optimization algorithms. Platforms such as IOHprofiler [DWY+18] and COCO [HAM+16] provide standardized environments for comparing iterative optimization heuristics. These platforms offer a suite of benchmark functions and performance metrics, facilitating comprehensive assessments of algorithm efficiency, robustness, and convergence speed.
In our benchmarking study, we compare the IWO algorithm to the following established baselines: Genetic Algorithm (GA) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) in addition to Random Search (RS). Additionally, we discuss SA, since it has some similarities with the IWO algorithm.
Genetic Algorithms: Developed by John Holland in the 1970s [Hol75], are among the most widely used nature-inspired optimization techniques. GAs simulate the process of natural selection, using operators such as selection, crossover, and mutation to evolve solutions over generations.
Covariance Matrix Adaptation Evolution Strategy: Developed by Hansen and Ostermeier in 2001 [HO01], is known for its robust performance on non-separable and ill-conditioned problems. Extensions like the augmented CMA-ES (a-CMA-ES) and Bimodal Population CMA-ES (BIPOP-CMA-ES) further enhance its adaptability and performance.
Simulated Annealing: Introduced by Kirkpatrick, Gelatt, and Vecchi in 1983 [KGV83], is inspired by the annealing process in metallurgy. SA explores the search space by probabilistically accepting worse solutions to escape local optima, gradually reducing the acceptance probability as the algorithm progresses. This method shares similarities with IWO in terms of balancing exploration and exploitation, but differs in its single-solution focus compared to IWO's population-based approach.
Particle Swarm Optimization (PSO): Developed by Eberhart and Kennedy in 1995 [EK95], Particle Swarm Optimization is based on the social behavior of birds and fish. Which optimizes a problem by iteratively improving a population of candidate solutions, called particles, based on their own experience and the experience of neighboring particles.

3 Algorithm Description

3.1 Problem definition

The IWO algorithm was developed to solve complex numerical optimization problems, which are characterized by the need of finding optimal or near-optimal solutions in large, multidimensional search spaces. These kinds of issues are common in many scientific, engineering, and economic domains, where exact solutions can greatly improve productivity and performance. Traditional optimization methods often struggle with the scale and complexity of these problems, leading to suboptimal performance or insufficient computation times. The IWO method aims to effectively explore the search space and converge on optimal solutions.

3.2 Input and Output specification

The IWO algorithm relies on a set of precisely defined inputs in order to initiate and execute the optimization process. These inputs are essential since they not only initiate the process but also impact the algorithm's performance through various search spaces.
To adjust the algorithm's performance to specific optimization problems, the user needs to explicitly specify the following parameters (inputs):

- **Nonlinear modulation index (n):** Modulates dispersal and reproduction rates nonlinearly across generations.
- **Initial sigma:** The initial value of standard deviation for seed dispersal.
- **Final sigma:** The final value of standard deviation for seed dispersal.
- **Minimum seeds:** The lowest number of seeds any weed can produce.
- **Maximum seeds:** The highest number of seeds the fittest weeds can produce.
- **Number of generations:** The total number of iterations the algorithm will execute.
- **Initial population size:** The number of initial solutions (weeds).
- **Maximum population size:** The upper limit of the number of weeds in the population.

After completing its run, the IWO algorithm provides main outputs that are critical for evaluating its effectiveness:

- **Optimal solution:** The best solution found during the generations (iterations), representing the most optimal or near-optimal solution to the problem.
- **Fitness value:** The evaluation score of the optimal solution, indicating how well it satisfies the problem's objectives.

3.3 Algorithm phases

The IWO algorithm progresses through the following phases:

Initialization

This phase involves the creation of an initial population of seeds (solutions), randomly distributed over the d-dimensional search space.

Reproduction

In this phase, each plant in the population is allowed to produce seeds based on its fitness relative to the colony's overall range of fitness. In other words, the number of seeds a plant produces scales linearly with its fitness, from a predefined minimum to a maximum. This relationship is illustrated in Figure 1. Such a scaling mechanism ensures that the seed production incrementally increases from the least fit to the most fit plants of the population.
This method of reproduction adds a crucial aspect to the search algorithm. It is a common assumption in traditional evolutionary algorithms that individuals with higher fitness have a higher chance of surviving and reproducing, while those who have lower fitness are usually eliminated. As noted by Yuchi and Kim[YK05], this viewpoint, however, ignores a critical feature of evolutionary algorithms: their probabilistic and recurrent nature. Sometimes, individuals that initially seem less feasible may carry valuable information that can significantly contribute to the evolutionary process. Additionally, traversing infeasible regions of the search space can sometimes facilitate reaching optimal solutions more efficiently, especially in non-convex environments.
As a result, this strategy gives infeasible plants the opportunity to survive and reproduce, similar to natural systems. In this way, the algorithm improves the search process by utilizing the information that potentially exists within infeasible plants.

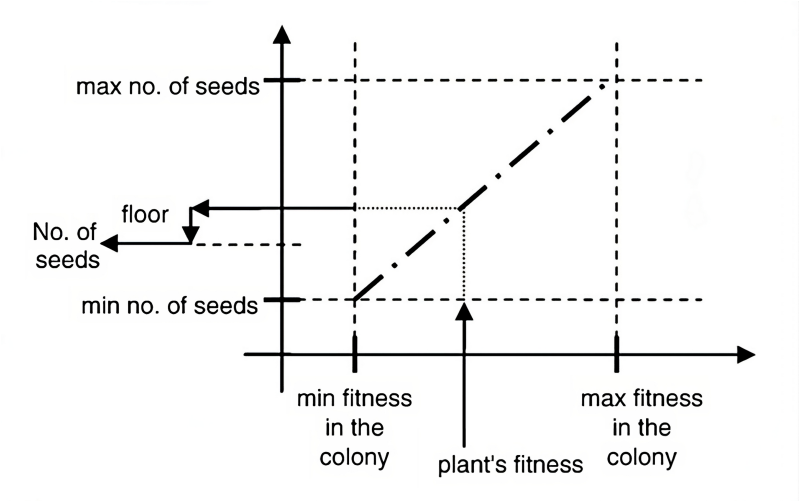

Figure 1: Relationship between plant fitness and seed production in the IWO algorithm. Plants with higher fitness produce more seeds, with the number of seeds ranging from a minimum to a maximum value [ML06].

Spatial dispersal

In this phase, randomness and adaptability of the algorithm are introduced where newly generated seeds are randomly distributed across the `d`-dimensional search space using normally distributed random numbers with a mean of zero and a varying variance: $x \sim N(0, \sigma)$. This results in seeds being dispersed close to their parent plant.
The standard deviation σ of the random distribution gradually reduces from an initial value σ_{initial} to a final value σ_{final} with each generation. As the algorithm progresses, this gradual reduction narrows the search, focusing more on promising regions.
The nonlinear reduction of the standard deviation is shown in Equation (1) [ML06]:

$$\sigma_{\text{gen}} = \frac{(\text{gen}_{\max} - \text{gen})^n}{(\text{gen}_{\max})^n}(\sigma_{\text{initial}} - \sigma_{\text{final}}) + \sigma_{\text{final}} \tag{1}$$

Where:

- $\mathbf{gen_{max}}$ is the maximum number of generations (iterations).
- $\sigma_{\mathbf{gen}}$ is the standard deviation at the present time step (current generation).
- `n` is the nonlinear modulation index.

This formulation adjusts the seed dispersal mechanism such that the likelihood of seeds being dropped in distant locations decreases nonlinearly with each generation. This strategy results in grouping fitter plants while less suitable ones are gradually eliminated.

Competitive exclusion

Finally, a selection mechanism is applied simulating ecological systems, where plant populations are governed by the need for competition to prevent uncontrolled growth. Similarly, as the algorithm iterates, rapid reproduction causes the number of plants in a colony to reach a predefined maximum number of individuals in the population. At this point, it is generally observed that fitter plants have reproduced more successfully than the less desirable ones. Once the upper limit is reached, an elimination mechanism is activated to maintain population balance and enhance the quality of the colony.
This works by first allowing each weed to produce seeds, as detailed in Section 3.3. These seeds then spread over the search space, as explained in Section 3.3. When all seeds have found their position, both seeds and their parent plants are together ranked by fitness. Then weeds with low fitness are removed until the population reaches the limit, see Figure 2. This method of selection guarantees that only the fittest individuals survive and grow, resulting in a strong and competitive colony. As explained in Section 3.3, this process permits plants with lower fitness to reproduce, and if their offspring have good enough fitness in the colony, they will survive.

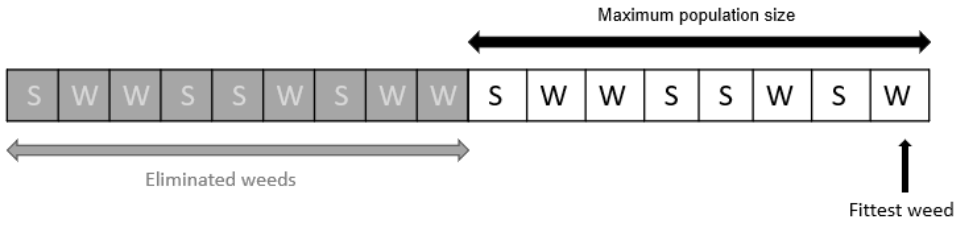

Figure 2: Competitive exclusion process in IWO algorithm: (S) are seeds, (W) are parent weeds. This illustration shows how less fit weeds (grey boxes) are eliminated to maintain the maximum population size. The fittest weeds (white boxes) survive and reproduce, ensuring that only the strongest individuals remain in the population, thereby improving overall colony fitness.

3.4 Pseudocode

The IWO pseudocode is shown in Algorithm 1, to illustrate the sequence of steps performed by the algorithm, encapsulating its core logic in a clear way.

Algorithm 1 Invasive Weed Optimization

for $i = 1 \rightarrow M_0$ **do**
  $f_i \leftarrow f(\mathbf{x}_i), \quad \mathbf{x}_i \leftarrow \boldsymbol{\mathcal{U}}(\mathbf{x}^{\min}, \mathbf{x}^{\max})$ ▷ Initialize
end for
for $g = 1 \rightarrow G$ **do**
  $f, X \leftarrow \text{sort}(f, X)$
  $n_{\text{seeds}} \leftarrow \text{interp}(f, [f_{\min}, f_{\max}], [S_{\max}, S_{\min}])$ ▷ Reproduction
  Calculate σ according to Equation (1)
  for $i = 1 \rightarrow M$ **do**
    for $j = 1 \rightarrow n_{\text{seeds}}[i]$ **do**
      $s_j \leftarrow \boldsymbol{\mathcal{N}}(\mathbf{x}_i, \sigma)$
      $fs_j \leftarrow f(s_j)$ ▷ Spatial dispersal
    end for
    $X_{\text{new}} \leftarrow \text{concatenate}(X, Xs)$
    $f_{\text{new}} \leftarrow \text{concatenate}(f, fs)$
  end for
  $f, X \leftarrow \text{sort}(f_{\text{new}}, X_{\text{new}})$ ▷ Competitive exclusion
  $X \leftarrow X[: M_{\max}]$
  $f \leftarrow f[: M_{\max}]$
end for

Definitions:
The following symbols are used throughout the pseudocode to represent key parameters and variables of the IWO algorithm:

- M: Total number of individuals (weeds) in the population.
- $M_{\max}$: Maximum allowable size of the population.
- M_0: Initial population size.
- $\mathbf{x}_i$: Position or solution vector for the i-th individual in the population.
- f_i: Fitness of the i-th individual.
- $\mathbf{x}^{\min}, \mathbf{x}^{\max}$: Minimum and maximum bounds for the solution space.
- G: Total number of generations the algorithm will run.
- n_{seeds}: Number of seeds produced by each individual.
- $S_{\max}, S_{\min}$: Maximum and minimum number of seeds that can be produced by the fittest and least fit individuals, respectively.
- σ: Standard deviation used in the normal distribution.
- $\mathbf{s}_j$: Position of the j-th seed produced by an individual.
- fs_j: Fitness of the j-th seed.
- $X_{\text{new}}, f_{\text{new}}$: Temporary containers for newly generated seeds and their fitness values before being merged back into the main population.

3.5 Edge cases and limitations

3.5.1 Nonlinear reduction equation

Using the original equation of the nonlinear reduction of the standard deviation as presented in the paper (Equation (1)), the nonlinear modulation index (n) is constrained to a maximum values up to 9 when maximum generation number ($\text{gen}_{\max}$) is set to 100. This limitation arises because $(\text{gen}_{\max})^n$ results in exceedingly large numbers. It is impractical to evaluate higher values of n.
To address this issue, we propose a modification where n can be applied to the entire fraction as shown in Equation (2):

$$\sigma_{\text{gen}} = \left(\frac{\text{gen}_{\max} - \text{gen}}{\text{gen}_{\max}} \right)^n (\sigma_{\text{initial}} - \sigma_{\text{final}}) + \sigma_{\text{final}} \tag{2}$$

This modification allows for testing significantly larger n values, even up to 1000, if desired.
Optimal performance for some test functions may be achieved with relatively large $\text{gen}_{\max}$ values (e.g., 800 as referenced in the paper) and high values of n. The original study only explored n values up to a maximum of 3, with $\text{gen}_{\max}$ set at 800.

As we can see in Figure 3, employing values of $n > 9$, in the original equation, with the maximum number of generations $\text{gen}_{\max}$ set to 800 results in abnormal values for σ. In contrast, the modified equation yields correct and stable values for σ even for significantly larger values of n. This enhances the robustness and applicability of the algorithm for a broader range of n values.

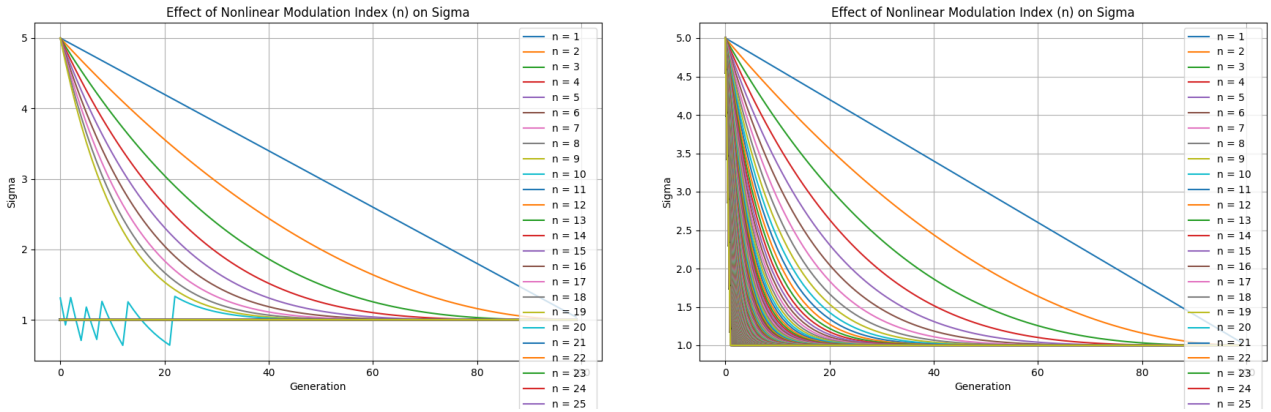

Figure 3: Effect of Nonlinear modulation index (n) on σ Using the original equation Equation (1) (left) and the modified equation Equation (2) (right).

3.5.2 Maximum number of generations

The performance of the IWO algorithm is significantly influenced by the number of generations. This is because the transition from exploring the search space to exploiting the best-found solutions is based upon the current generation number, as described in Equation (1). For instance, with a limited computational budget, setting the maximum number of generations to a specific number (e.g., 100), which allows the algorithm to have sufficient time (i.e., number of function evaluations) to reach the exploitation phase, would make the algorithm more powerful than with the same budget but a maximum number of generations set to a higher number (e.g., 1000). The algorithm then lacks the necessary time to transition into the exploitation phase, resulting in more stochastic performance.
However, arbitrarily choosing a large evaluation budget is not always possible due to time and resource constraints. Therefore, it is crucial to select the maximum number of generations properly to ensure that the algorithm maximizes the number of function evaluations used within the available budget.

3.6 Implementation

The implementation of the IWO) algorithm was carried out using Python, utilizing the IOHprofiler framework for benchmarking and performance analysis. The development environment was based on Python 3, with essential libraries including:

- **ioh:** For problem interfacing and collecting analytical data[1].
- **numpy:** For numerical operations[2].
- **sys:** For handling command-line arguments[3].

[1] https://github.com/IOHprofiler/IOHexperimenter
[2] https://numpy.org/
[3] https://docs.python.org/3/library/sys.html

The core of the implementation is the `IWO` class , which encapsulates all the functionalities required to execute the optimization algorithm (see Appendix A.1 for more details). This class manages the initialization of parameters, reproduction of solutions, spatial dispersal, and competitive exclusion phases of the IWO algorithm.
The experiments were designed to test the algorithm on 24 benchmark functions from the BBOB suite, across various dimensions and problem instances. The data were collected using IOHexperimenter then post processed using IOHanalyzer, providing insights into the algorithm's performance relative to other optimization methods.

3.7 Originality:

3.7.1 Biological inspiration

The IWO algorithm is inspired by the colonization behavior of weeds, which differentiates it from other optimization algorithms.
While GAs are based on genetic evolution, SA is based on the annealing process in metallurgy and PSO mimics the social behavior of birds or fish, IWO models the adaptive and reproductive strategies of weeds, making it a novel approach in optimization.

3.7.2 Comparison with Simulated Annealing

Although IWO operates on a population of solutions, whereas SA typically improves a single solution iteratively, IWO has some similarities with SA in terms of:

- **Exploration to exploitation transition:** The transition from exploration to exploitation in SA is controlled by a temperature parameter. SA starts with a high temperature ($\max_T$), allowing a high probability of accepting worse solutions, encouraging exploration. As the temperature decreases to a minimum ($\min_T$), the probability of accepting worse solutions decreases, focusing on exploitation. The temperature decreases according to a cooling schedule, which is based on the current iteration number. The cooling schedule can be linear, exponential, or based on other functions.
  In IWO the transition from exploration to exploitation is determined by the initial and final value of the standard deviation (σ). Initially, seeds are dispersed far from the parent plant (high σ), representing exploration. As the generations progress, the dispersion radius decreases (low σ), focusing on exploiting the best-known solutions. The transition from $\sigma_{initial}$ to σ_{final} depends on the nonlinear modulation index (n), maximum generation number and the current generation number (iteration number).

- **Mechanisms of search:** SA generates new solutions through random moves near the current solution, whereas the IWO algorithm disperses seeds randomly around parent plants.
  When IWO is configured with parameter values: $S_{\max} = 1$, $S_{\min} = 0$ and $M_{\max} = 1$, its behavior closely mirrors that of SA. However, important distinctions and similarities exist between the two algorithms in terms of their operational mechanisms and conceptual foundations.

Although the IWO shares some similarities with SA, such as the transition from exploration to exploitation and the use of stochastic processes, its biological inspiration and distinctive mechanisms make it unique.

4 Experiments

In this section, various experiments were conducted to evaluate the performance of the IWO algorithm. The experiments are designed to achieve the following objectives:

- Verify the correctness of the IWO algorithm implementation.
- Reproduce the experiments and validate the findings of the original paper using the same parameter settings and experimental conditions.
- Explore the effect of varying key parameters on the performance of the IWO algorithm.
- Compare the performance of the IWO algorithm against GA, a-CMA-ES, BIPOP-CMA-ES, and a RS.

4.1 BBOB functions

The performance of the IWO algorithm was evaluated using the 24 continues and noise-free BBOB functions [HAM+16]. These functions are categorized into several groups based on their characteristics, which provide a comprehensive assessment of optimization algorithms. Each category tests different aspects of the algorithm's performance. The categories are:

- **Separable functions:** These functions can be optimized by solving smaller, independent subproblems. Examples: Sphere and Linear Slope functions.
- **Low or moderately conditioned functions:** These functions have a moderate condition number, making them moderately difficult to optimize. Examples: Rosenbrock and Attractive Sector functions.
- **Highly conditioned functions:** These functions have a high condition number, posing a significant challenge for optimization algorithms. Examples: Discus and Bent Cigar functions.
- **Multi-modal functions with adequate global structure:** These functions have multiple local optima but a clear global structure. Examples: Schaffer's F7 and Weierstrass functions.
- **Multi-modal functions with weak global structure:** These functions have a complex landscape with many local optima and a less clear global structure. Examples: Schwefel and Katsuura functions.

4.2 Implementation correctness

To ensure the correctness of the IWO implementation, the algorithm's performance is evaluated on the Sphere function in a 2-dimensional search space using the IOHprofiler to verify that the fitness values improve as expected over generations. Additionally, visualizations were created using `matplotlib` to observe how the population evolves and progresses across generations.

The results in Figures 4, 5 , collectively validate the correctness of the IWO algorithm implementation. The consistent improvement in fitness values over generations, as observed in IOHprofiler results, confirms that the algorithm is functioning as expected. Additionally, the visualizations provide a clear depiction of the population's evolutionary process, further supporting the algorithm's correctness and its ability to effectively explore and exploit the search space.

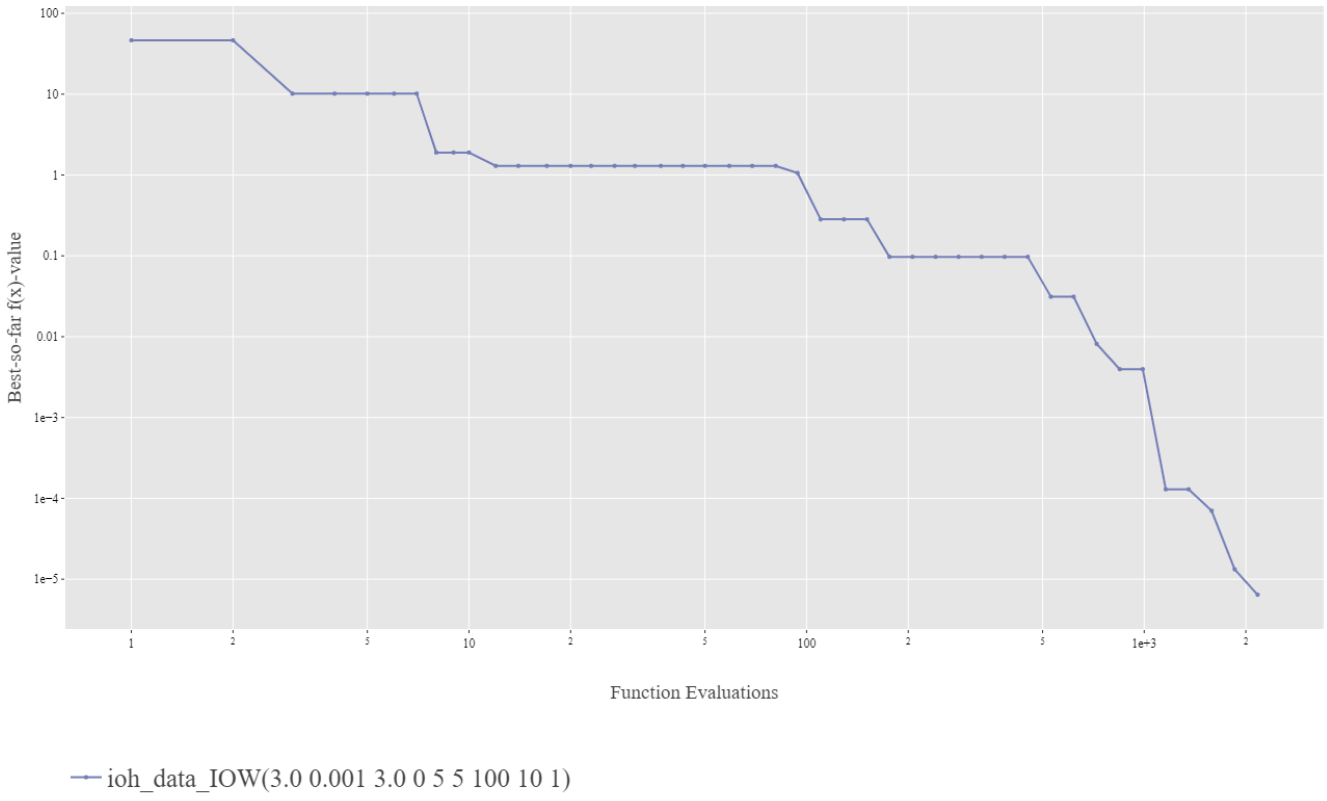

Figure 4: Fitness value progression of the IWO algorithm on Sphere function using IOHprofiler. The plots show the best-so-far fitness value over the number of function evaluations.

4.3 Original experiments

In the original IWO paper [ML06], the authors evaluate the performance of the proposed algorithm based on the following criteria:

1. **Convergence:** The algorithm's ability to find the global minimum is tested for three widely used benchmark functions: Sphere, Griewank, and Rastrigin functions. In order to show that the algorithm consistently finds global solutions, these findings are compared to those produced using a standard GA.

2. **Effects of tuning parameters:** The IWO algorithm is then tested with a high-dimensional instance of the Rastrigin function with a dimensionality of `d=30` with `100` runs in each parameter settings to show how the IWO algorithm's parameter tweaking affects its performance.

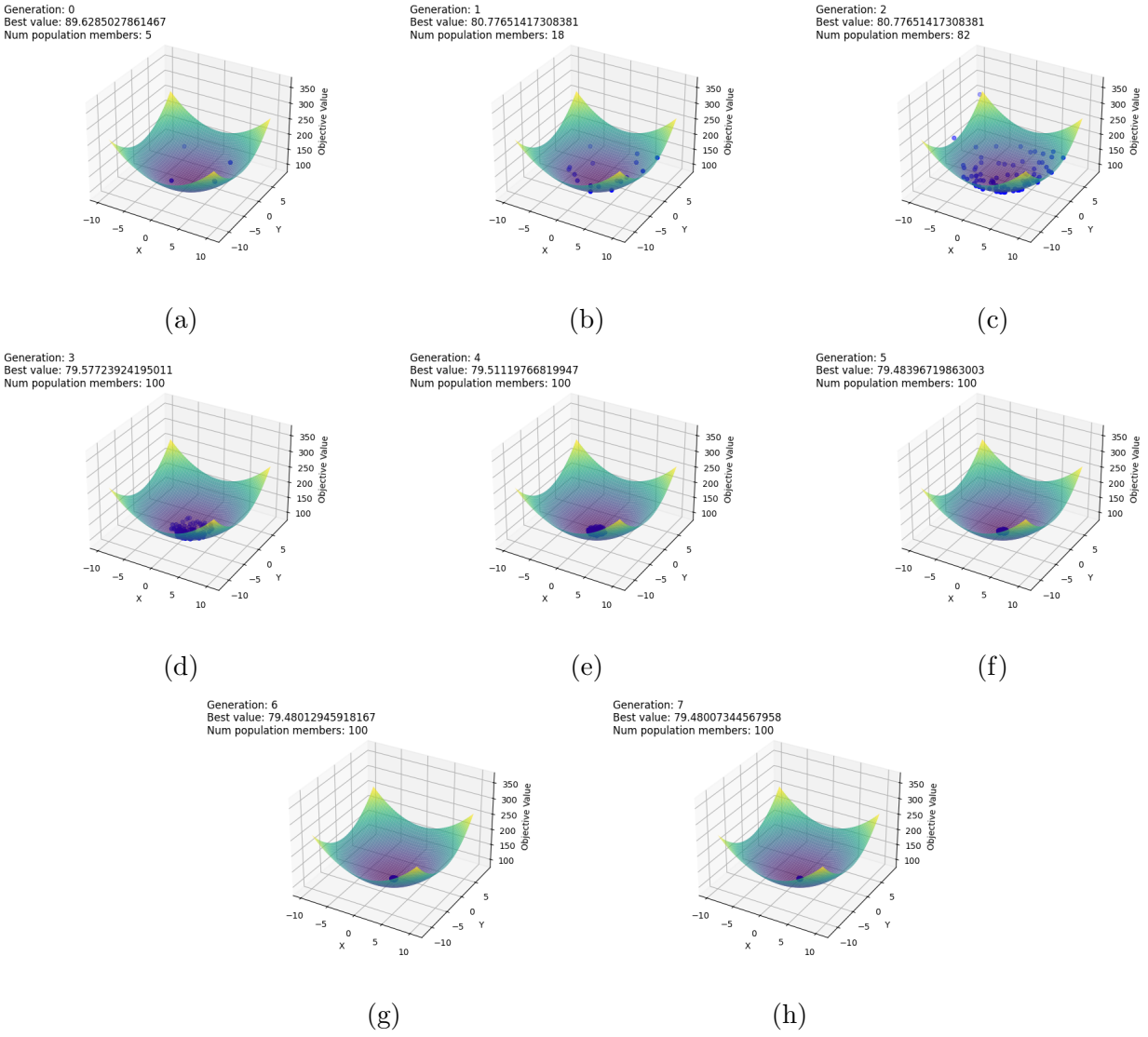

Figure 5: This figure illustrates the evolution of the population over eight generations of the IWO algorithm. Each subplot shows the best value found, the number of population members, and a 3D representation of the solution space at each generation, highlighting how the algorithm converges towards the optimal solution.

3. **Comparison with GAs, MAs, PSO, and SFL:** The IWO algorithm is compared to other commonly used numerical optimization methods such as: GAs, PSO, Memetic Algorithms, and Shuffled Frog Leaping. This comparison is performed by optimizing an extremely high-order Griewank function with dimensions `d=10`, `20`, `50`, and `100`, in addition to the EF10 function.

4. **Comparison with SDS, SSA, and DSSA:** Because of some similarities between the proposed algorithm and SA method, the IWO algorithm is compared to several types of simulated annealing, particularly Simplex Simulated Annealing and Direct Search Simulated Annealing. This evaluation is carried out using the Easom and Griewank functions.

4.4 Validation of original findings

To validate and extend these findings, the same experiments were performed using the IOHexperimenter tool. This involved running the IWO algorithm on the functions used in the original paper that are also included in the BBOB suite and comparing its results with the ones in the original paper.

4.4.1 Convergence

The final fitness value observed in the original paper tested on the Sphere function was: `2.4362e-8`, with dimension `d=2`, where the optimal value is: `0.0`.
Using the same parameter settings Table 1, the results in Figure 6, are obtained using IOHanalyzer:

	ID	DIM	funcId	runs	best reached	succ	budget	max evals used	worst recorded	worst reached	mean reached	median reached
	All	All	All	All	All	All	All	All	All	All	All	All
1	ioh_data_IOW(3.0 0.001 3.0 0 5 10 15 100 0)	2	1	100	2e-10	1	4126	4062	135.24	5.2e-8	1.1e-8	6e-9

Figure 6: IWO algorithm performance on the Sphere function with `d=2`.

We can see in Figure 6, that the mean value reached is `1.1e-8` which is even better than the one observed in the original paper. Which means that we accept the claim of the original paper about the performance of the IWO on the Sphere function with dimension `d=2`.

Symbol	Description	Value
M_0	Initial population size	10
M_{max}	Maximum population size	15
G	Total number of generations	100
d	Problem dimension	2
S_{max}	Maximum number of seeds	5
S_{min}	Minimum number of seeds	0
n	Nonlinear modulation index	3
$\sigma_{initial}$	Initial value of standard deviation	3
σ_{final}	Final value of standard deviation	0.001

Table 1: IWO parameter values for Sphere function minimization

Another challenging optimization problem is addressed to demonstrate the abilities of the IWO, which is minimization of the Rastrigin function with dimension `d=2`, using parameter values as shown in Table 2. The performance of the IWO is shown in the paper using a fixed-budget plot against the GA, where the budget is: `500` iterations (see Figure 7).

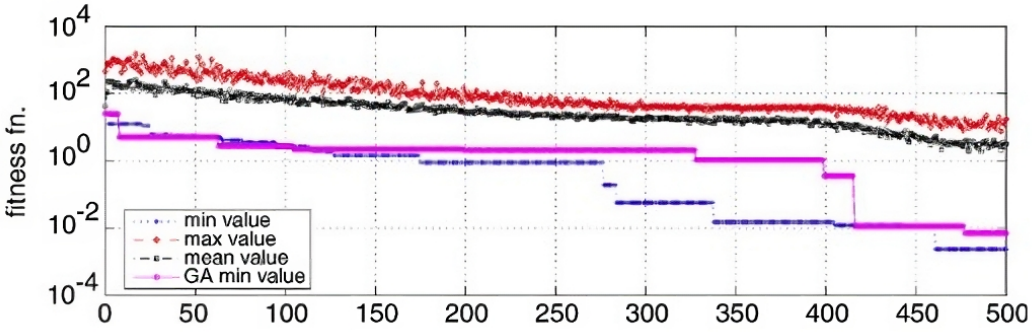

Figure 7: IWO algorithm performance on the Rastrigin function with `d=2` and budget `500` iterations [ML06].

Symbol	Description	Value
M_0	Initial population size	10
M_{max}	Maximum population size	30
G	Total number of generations	500
d	Problem dimension	2
S_{max}	Maximum number of seeds	5
S_{min}	Minimum number of seeds	0
n	Nonlinear modulation index	3
$\sigma_{initial}$	Initial value of standard deviation	10
σ_{final}	Final value of standard deviation	0.1

Table 2: IWO parameter values for Rastrigin function minimization

In Figure 7, we see that the minimum fitness value achieved by the IWO is consistently lower than that of GA after certain iterations, suggesting that IWO might be more effective for this specific problem within the given iterations.

To validate and compare the performance of the IWO with GA, I replicated the experiment using the same parameter values as specified in Table 2, on the same benchmark function. Figure 8 illustrates the results obtained using IOHanalyzer.
As illustrated in Figure 8, the mean fitness values achieved by the IWO algorithm were consistently higher than those achieved by the GA, indicating that, on average, the GA outperformed the IWO under the chosen experimental settings.
However, considering that the original paper's focus was on minimum values, it is plausible that the IWO algorithm may outperform the GA in certain runs in terms of finding the best solutions, despite its lower average performance.

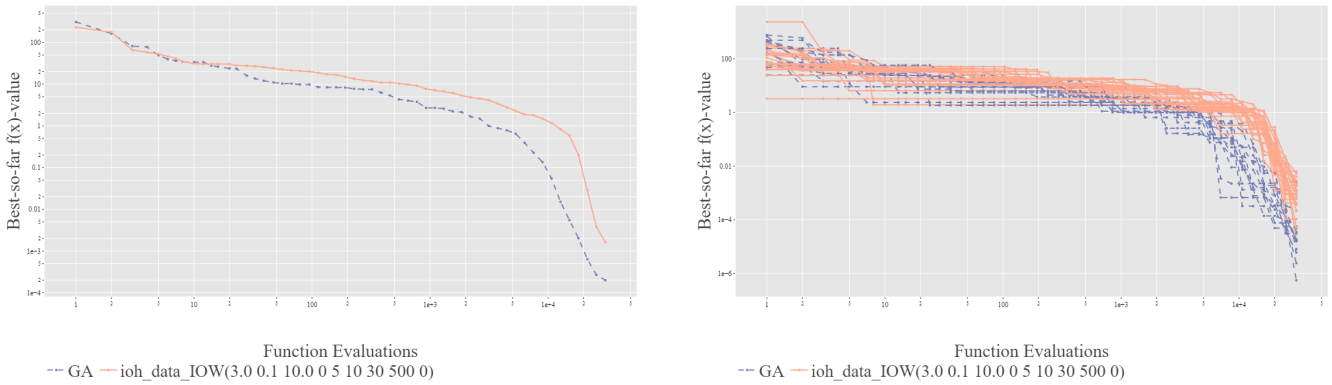

Figure 8: IWO algorithm performance against GA on the Rastrigin function with `d=2` and budget `500` iterations which is about `30512` function evaluations. Left figure shows the mean value, and the right figure shows the individual runs.

4.4.2 Effects of tuning parameters

The IWO algorithm is then tested with a high-dimensional instance of the Rastrigin function with a dimensionality of `d = 30` with `100` runs in each parameter settings, see Table 3.
The presented results of the original paper in Figure 9 shows how the IWO algorithm's parameter tweaking affects its performance by looking at:

- the percentage of success, as represented by the number of trials required for the object function to reach its known target values, which is equal or lower than `0.05` in this experiment. [ML06]

- the average value of the solution obtained in all trails. [ML06]

In all experiments, the search is stopped when the maximum number of generations (iterations) is reached.
When replicating these experiments using the IOHprofiler with the same settings as specified in Table 3, the results present a different picture.
In Figures 10 and 11, we can see that Increasing the values of the parameters n, M_{max} and G separately, each consistently leads to lower mean values. This indicates that enhancing these parameters individually improves the algorithm's performance. This observation contrasts with the findings of the original paper, where, under the experimental settings ($n = 3$, $G = 100$, $M_{max} = 20, 40,$ and 60), the results indicated a decline in performance with an increase in population size.
However, the success rate values reported in the original paper do not align with the outcomes observed in the replicated experiments neither with the mean values reported in the original paper. The definition of the success rate is ambiguous. The description states, "the percentage of success, as represented by the number of trials **required** for the objective function to reach its known target values, which is equal or lower than **0.05** in this experiment," [ML06] suggesting that a lower percentage indicates better performance since fewer runs are needed to reach the desired target. However, the results in Figure 9 and their interpretation in the original paper consider the success rate as the number of runs in which the algorithm **reached** the desired target value.

If the success rate indeed represents the number of runs reaching the target, it is confusing that using $G = 500$, $n = 3$, and $M_{max} = 60$ results in a mean value of `62.2004`, while using $G = 100$ with the same $n = 3$, and $M_{max} = 60$ results in a mean value of `1617.7`, despite the success rate for $G = 100$ being `67`, which is significantly higher than the success rate for $G = 500$ which is `5`. In both cases the reached mean value is substantially distant from the desired target of `0.05`.

Table 5 – Simulation results of high-dimensional Rastrigin function optimization

Max. population of weeds	Nonlinear modulation Index (n)	Max. no. of iterations	Comparison criteria	
			% Success	Mean solution
20			14	90.4242
40	3	500	12	69.3683
60			5	62.2004
20			27	2574.2
40	1	100	14	2427.1
60			9	2368.3
20			93	494.23
40	3	100	72	1538.7
60			67	1617.7
	1		11	230.74
20	2	500	26	92.957
	3		20	85.76

Figure 9: IWO algorithm performance on the Rastrigin function with dimensions `d=30` [ML06].

Symbol	Description	Value
M_0	Initial population size	10
M_{max}	Maximum population size	20, 40 or 60
G	Total number of generations	100 or 500
d	Problem dimension	30
S_{max}	Maximum number of seeds	3
S_{min}	Minimum number of seeds	0
n	Nonlinear modulation index	1, 2 or 3
$\sigma_{initial}$	Initial value of standard deviation	10
σ_{final}	Final value of standard deviation	0.02

Table 3: IWO parameter values for Rastrigin function minimization with dimensions `d=30`.

Using identical parameter settings, the best value obtained across all parameter settings during the replicated experiments was `39.71`, with the best mean solution value being `83.26`. These results are substantially distant from the desired target of `0.05`, indicating a considerable deviation from the original findings.

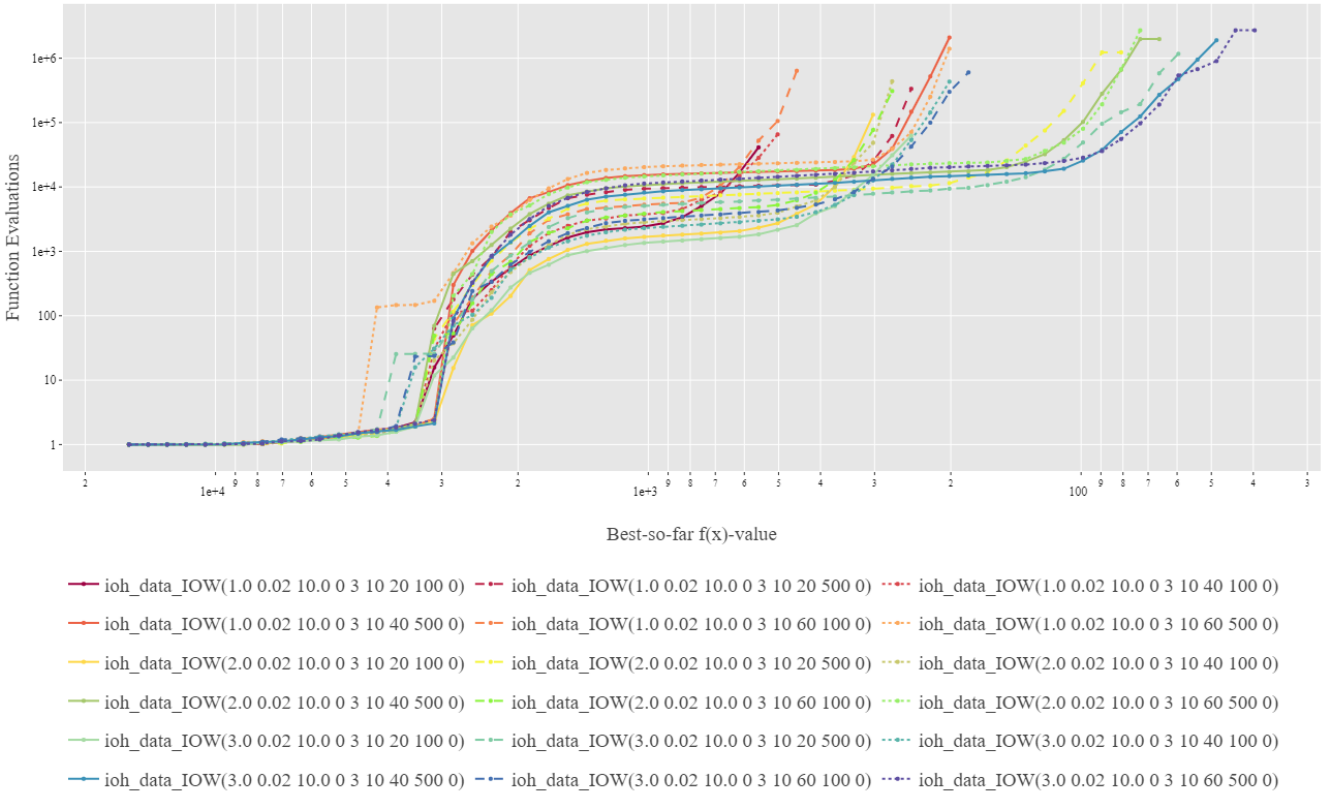

Figure 10: IWO algorithm performance on the Rastrigin function with dimensions `d=30` and various parameter settings.

	ID	DIM	funcId	runs	best reached	succ	budget	max evals used	worst recorded	worst reached	mean reached	median reached
	All	All	All	All	All	All	All	All	All	All	All	All
18	ioh_data_IOW(3.0 0.02 10.0 0 3 10 60 500 0)	30	3	100	39.71	1	27117	27013	10028.29	136.69	83.26	81.26
16	ioh_data_IOW(3.0 0.02 10.0 0 3 10 40 500 0)	30	3	100	47.77	1	18988	18849	9891.7	160.35	91.13	90.5
14	ioh_data_IOW(3.0 0.02 10.0 0 3 10 20 500 0)	30	3	100	59.6	1	11695	11669	15846.59	211.98	120.87	115.68
10	ioh_data_IOW(2.0 0.02 10.0 0 3 10 40 500 0)	30	3	100	65.82	1	19796	19657	8373.29	185.48	119.62	116.15
12	ioh_data_IOW(2.0 0.02 10.0 0 3 10 60 500 0)	30	3	100	68.34	1	27108	27041	11195.07	162.23	109.21	106.07
8	ioh_data_IOW(2.0 0.02 10.0 0 3 10 20 500 0)	30	3	100	74.89	1	12339	12318	8422.31	255.6	154.45	151.53
17	ioh_data_IOW(3.0 0.02 10.0 0 3 10 60 100 0)	30	3	100	178.15	1	6021	5994	10178.45	465.69	315.63	315.63
6	ioh_data_IOW(1.0 0.02 10.0 0 3 10 60 500 0)	30	3	100	191.45	1	28092	28015	10853.78	341.12	257.4	257.24
15	ioh_data_IOW(3.0 0.02 10.0 0 3 10 40 100 0)	30	3	100	195.92	1	4321	4282	10821.04	470.49	332.94	330.74
4	ioh_data_IOW(1.0 0.02 10.0 0 3 10 40 500 0)	30	3	100	200.12	1	20949	20883	9565.23	414.91	274.01	272.93
2	ioh_data_IOW(1.0 0.02 10.0 0 3 10 20 500 0)	30	3	100	231.98	1	13449	13432	10804.46	408.09	307.12	303.68
13	ioh_data_IOW(3.0 0.02 10.0 0 3 10 20 100 0)	30	3	100	237.14	1	2512	2495	8094.73	597.82	388.78	375.77
11	ioh_data_IOW(2.0 0.02 10.0 0 3 10 60 100 0)	30	3	100	262.05	1	6188	6129	12846.69	519.12	383.09	375.28
9	ioh_data_IOW(2.0 0.02 10.0 0 3 10 40 100 0)	30	3	100	266.97	1	4372	4341	10145.6	618.24	393.43	382.69
7	ioh_data_IOW(2.0 0.02 10.0 0 3 10 20 100 0)	30	3	100	294.23	1	2654	2643	10497.33	638.64	431.59	422.25
5	ioh_data_IOW(1.0 0.02 10.0 0 3 10 60 100 0)	30	3	100	448.98	1	6392	6338	10601.77	987.01	662.86	658.14
3	ioh_data_IOW(1.0 0.02 10.0 0 3 10 40 100 0)	30	3	100	469.59	1	4633	4610	14002.74	961.84	688.51	676.92
1	ioh_data_IOW(1.0 0.02 10.0 0 3 10 20 100 0)	30	3	100	506.19	1	2915	2905	9845.21	1166.4	745.25	742.84

Figure 11: IWO algorithm performance on the Rastrigin function with dimensions `d=30` and various parameter settings.

4.5 Performance optimization

In this section, we focus on optimizing the parameters of the IWO algorithm to enhance its performance across a diverse set of benchmarking scenarios.

Using IOHprofiler, these experiments were conducted on 24 functions from the BBOB benchmark suite [HAM+16], tested on instances `1-5`, for dimensions `2`, `5`, `20` and `40`, with `5` runs per instance to ensure statistical robustness.

The parameter values used in the experiments were varied and tested within specific ranges to evaluate their impact on the performance of the IWO algorithm. The default values, tested values, and the accepted range for each parameter are summarized in Table 7. To provide a standardized basis for these experiments, all subsequent parameter optimization trials were performed using the base setup values. The default values are taken from the original paper where the IWO algorithm is tested on the Sphere function with dimensions d=2.

To ensure a fair comparison in parameter optimization, it is essential to consider the following points:

- The number of function evaluations used in each parameter setting is significantly influenced by the parameters: number of generations, minimum number of seeds, maximum number of seeds, and population size.
- The performance of the IWO algorithm is heavily dependent on the number of generations. This is because the shift from exploring the search space to exploiting the best-found solutions depends on the current generation number. See Equation (1).

Therefore, comparing individual parameter settings that affect the number of function evaluations without adjusting the corresponding parameters is unfair.
By using Equation (3), we can benchmark the effect of the minimum seeds, maximum seeds, and population size by adjusting the number of generations accordingly. This adjustment ensures two key points:

- The algorithm uses roughly the same number of function evaluations across different parameter settings.
- The algorithm has sufficient time (i.e., function evaluations) to leverage both its exploration and exploitation capabilities.

$$N_{fe} = M_0 + \left(G \times \left(\frac{S_{\min} + S_{\max}}{2} \right) \times M_{\max} \right), \tag{3}$$

where:

- N_{fe}: Number of function evaluations
- $M_{\max}$: Maximum allowable size of the population.
- M_0: Initial population size.
- G: Total number of generations the algorithm will run.
- $S_{\max}, S_{\min}$: Maximum and minimum number of seeds that can be produced by the fittest and least fit individuals, respectively.

4.5.1 Minimum & Maximum seeds

Theoretically, higher values of the minimum seed number enhance the explorative capabilities of the algorithm by increasing the survival chances of weeds with lower fitness, leading to slower convergence towards the optimal solution. Conversely, lower values facilitate faster exploitation of the best-found solutions but increase the risk of becoming trapped in local optima. Similarly, higher values of the maximum seed number directly boost the number of seeds produced by high-fitness weeds, intensifying the search in promising areas and indirectly raising the survival chances of lower-fitness weeds by enlarging the overall seed values. On the other hand, lower maximum seed numbers may accelerate the exploitation of current best solutions but at the expense of diversity.

By calculating the number of function evaluations used in the basis parameter setting, we get N_{fe} = 3760. To maintain roughly the same number of function evaluations across other values of the minimum and maximum number of seeds, we adjust the generation number using the Equation (3). See Tables 4 and 5:

S_{min}	0	1	2	3	4	5
G	100	83	71	62	55	50

Table 4: Number of generations that should be used to maintain fair comparison between different values of the minimum seeds number.

S_{max}	1	2	3	4	5	6	7	8	9	10	20	50	100
G	500	250	166	125	100	83	71	62	55	50	25	10	5

Table 5: Number of generations that should be used to maintain fair comparison between different values of the maximum seeds number.

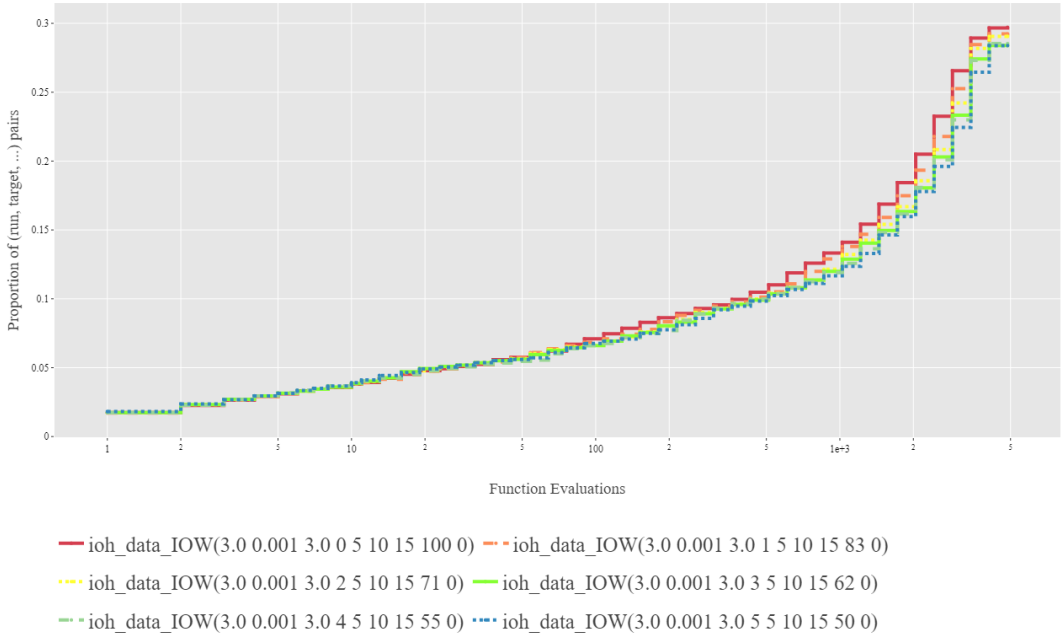

Figure 12: Aggregated ECDF across all functions and dimensions for different minimum seed values (fourth parameter in brackets) of the IWO algorithm, using BBOB spacing for the automatically generated ECDF-targets.

In Figures 12 and 13, we see how the IWO algorithm performs with different minimum seed values. We can see that configurations with lower minimum seed values (e.g., 0 and 1) generally outperform those with higher values (e.g., 4 and 5).

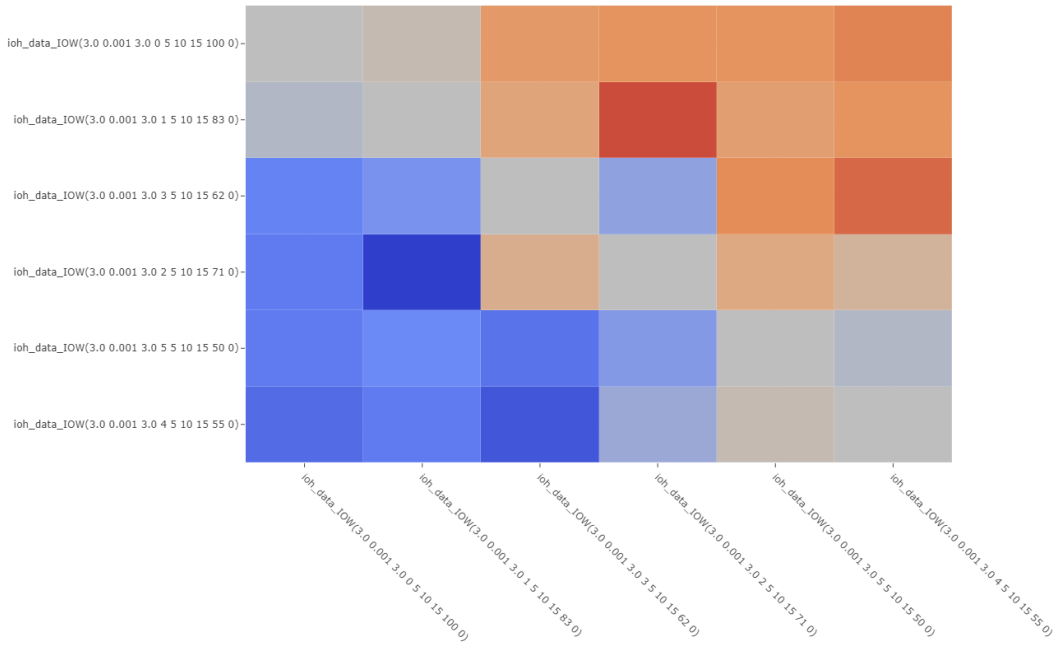

Figure 13: Heat map across all functions and dimensions for different minimum seed values (fourth parameter in brackets) of the IWO algorithm. Red means the algorithm on the y-axis outperforms the one on the x-axis.

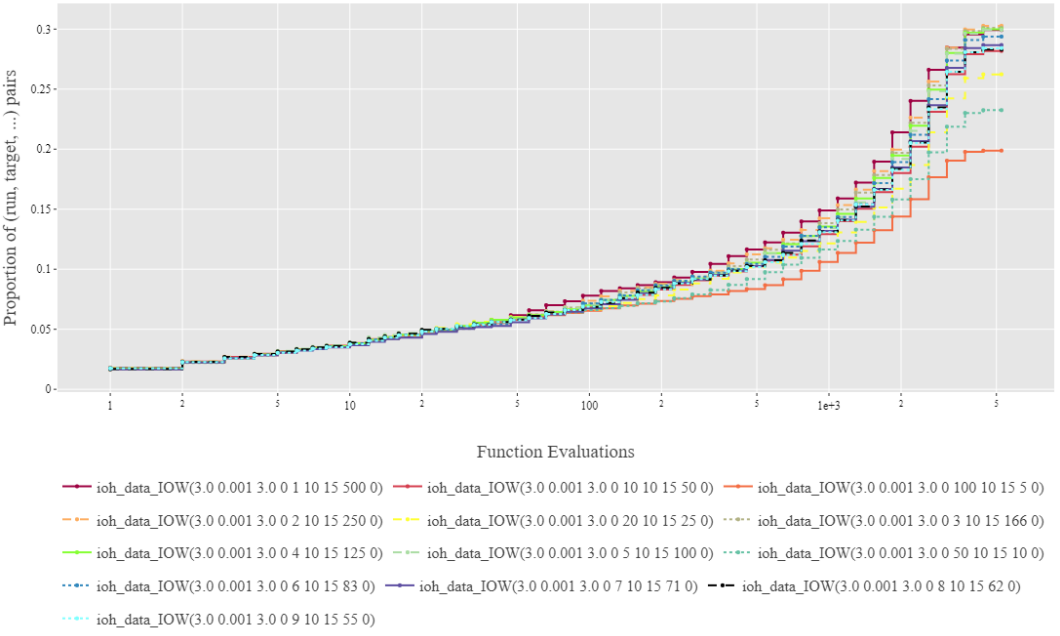

Figure 14: Aggregated ECDF across all functions and dimensions for different maximum seed values (fifth parameter in brackets) of the IWO algorithm, using BBOB spacing for the automatically generated ECDF-targets.

In Figures 14 and 15, we see how the IWO algorithm performs with different maximum seed values. We can see that configurations with lower maximum seed values (e.g., 1 – 5) generally outperform those with higher values (e.g., >10).

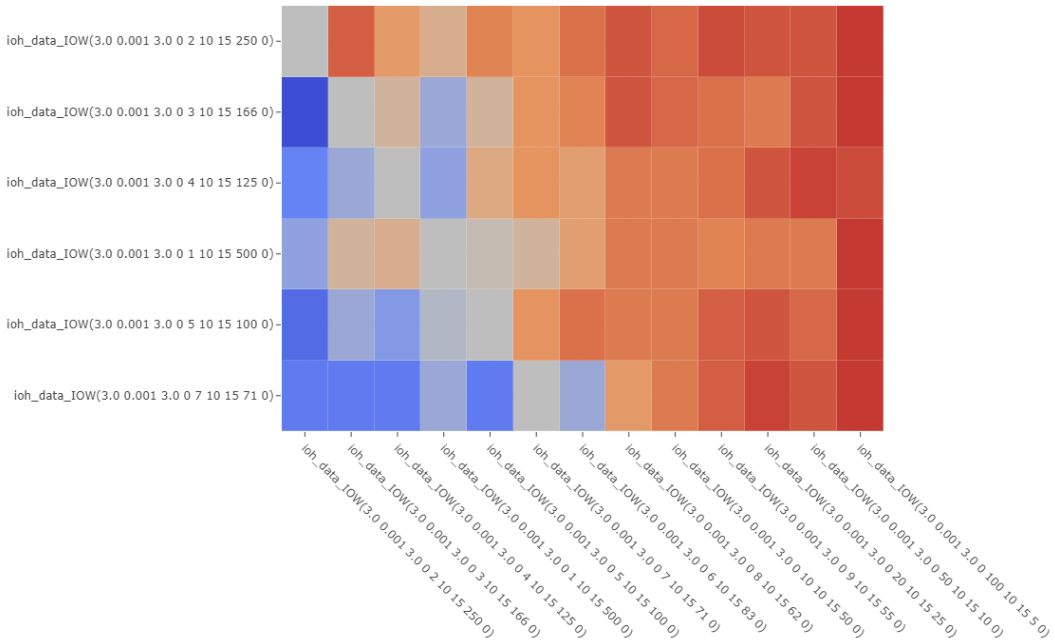

Figure 15: Heat map across all functions and dimensions for different maximum seed values (fifth parameter in brackets) of the IWO algorithm. Red means the algorithm on the y-axis outperforms the one on the x-axis.

4.5.2 Maximum population size

The maximum population size significantly impacts the diversity of the solutions. A larger population size enhances the algorithm's ability to explore a wider range of solutions, thereby increasing the likelihood of discovering the global optimum. However, this comes at the cost of a higher computational budget. For simpler problems, a smaller but sufficiently large population size is preferable to efficiently exploit the search space without excessive computational expense. Conversely, for more complex problems, larger population sizes are beneficial as they mitigate the risk of early convergence and help in avoiding local optima. Therefore, selecting an appropriate population size should be guided by the complexity of the problem.

To maintain roughly the same number of function evaluations across other values of the maximum population size, we adjust the generation number using the Equation (3). See Table 6.

M_{max}	10	15	20	30	50	80	100	200	500
G	150	100	75	50	30	18	15	8	3

Table 6: Number of generations that should be used to maintain fair comparison between different values of the maximum population size.

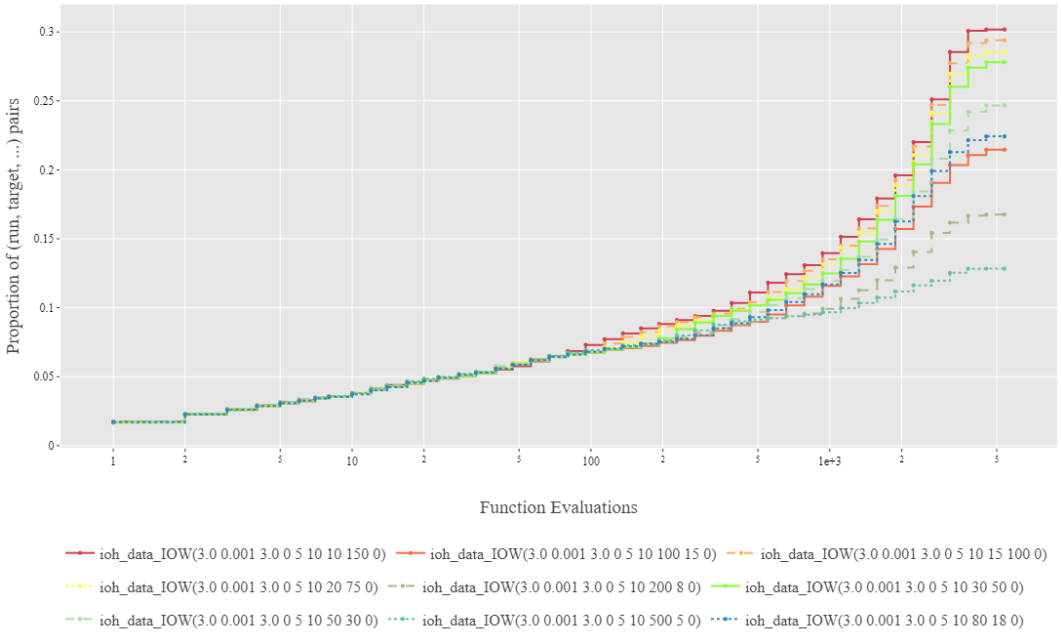

Figure 16: Aggregated ECDF across all functions and dimensions for different maximum population size values (seventh parameter in brackets) of the IWO algorithm, using BBOB spacing for the automatically generated ECDF-targets.

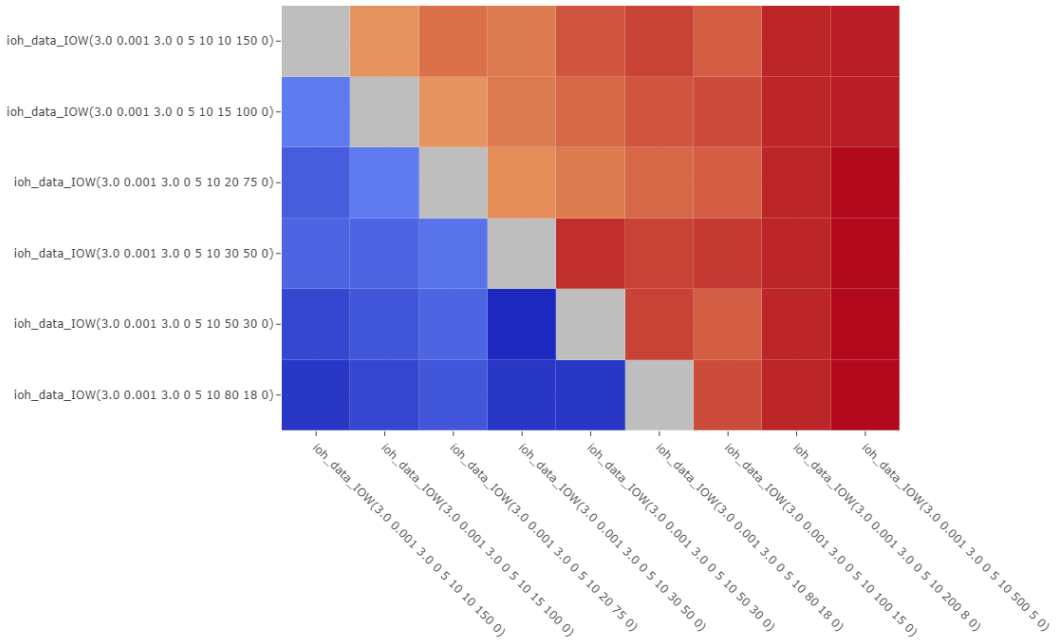

Figure 17: Heat map across all functions and dimensions for different maximum population size values (seventh parameter in brackets) of the IWO algorithm. Red means the algorithm on the y-axis outperforms the one on the x-axis.

In Figures 16 and 17, we see how the IWO algorithm performs with different values of the maximum population size. We can see that configurations with lower values (e.g., 10 – 20) generally outperform those with higher values (e.g., >30). This is because increasing the population size results in a higher number of function evaluations per generation, necessitating a reduction in the number of generations to remain within the specified budget. Consequently, this reduction in generations hinders the algorithm's ability to fully exploit the information from the best solutions, as is more effectively done with smaller population sizes.

4.5.3 Standard deviation & Nonlinear modulation index

The parameters σ_{initial}, σ_{final}, and n do not influence the number of function evaluations used by the algorithm. Therefore, there is no need to adjust other parameters to ensure a fair comparison between different settings.
These parameters primarily affect the transition speed from the exploration phase to the exploitation phase within the algorithm. Specifically, the σ_{initial} determines the dispersion radius of the seeds from the parent plant during the early generations, where the algorithm is focused on exploration. In contrast, the σ_{final} determines how close the seeds will be to their parent in later generations as the algorithm shifts towards exploitation. The nonlinear modulation index n governs the rate at which σ transitions from its initial to final value (see Figure 18).

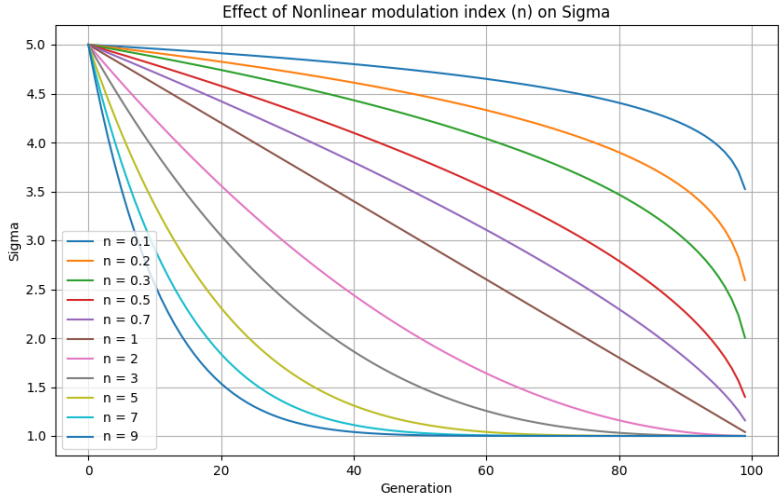

Figure 18: Effect of nonlinear modulation index (n) on standard deviation (σ). Higher n values leads to more exploitation than exploration (faster transition). Lower n values leads to more exploration than exploitation (slower transition).

In Figures 19 and 20, we see how the IWO algorithm performs with different values of the nonlinear modulation index (n). We can see that configurations with values (2 – 8) generally outperform those with higher (e.g., >10) and lower values (e.g., <1). This ensures an appropriate balance between exploration and exploitation.

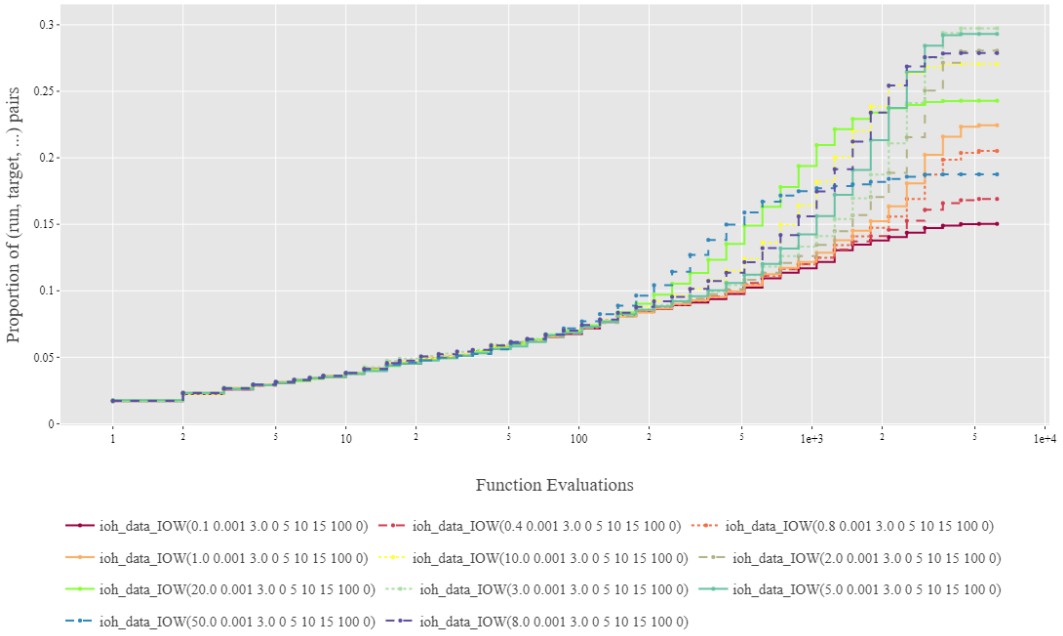

Figure 19: Aggregated ECDF across all functions and dimensions for different values of the nonlinear modulation index n (first parameter in brackets) of the IWO algorithm, using BBOB spacing for the automatically generated ECDF-targets.

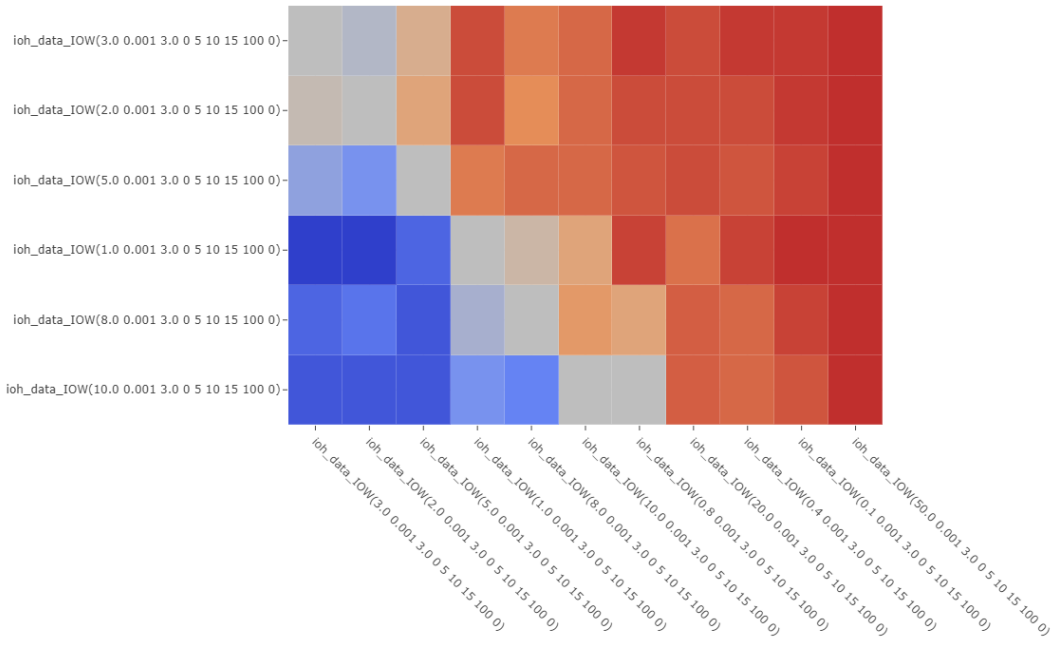

Figure 20: Heat map across all functions and dimensions for different values of the nonlinear modulation index n (first parameter in brackets) of the IWO algorithm. Red means the algorithm on the y-axis outperforms the one on the x-axis.

In Figures 21, 22, 23 and 24, we see how the IWO algorithm performs with different initial and final values of the standard deviation σ. We can see that configurations with values of ($1 \leq \sigma_{\text{initial}} \leq 5$) and (1e-5 $\leq \sigma_{\text{final}} \leq 0.01$), generally outperform those with higher values (e.g., σ_{initial} >10 and σ_{final} >0.1).

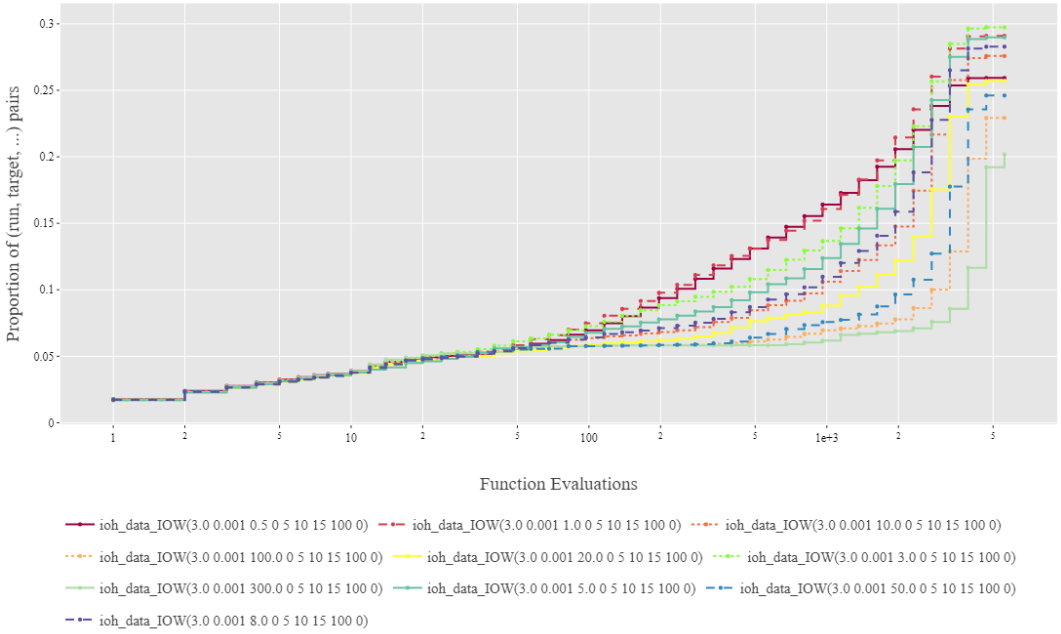

Figure 21: Aggregated ECDF across all functions and dimensions for different values of the σ_{initial} (third parameter in brackets) of the IWO algorithm, using BBOB spacing for the automatically generated ECDF-targets.

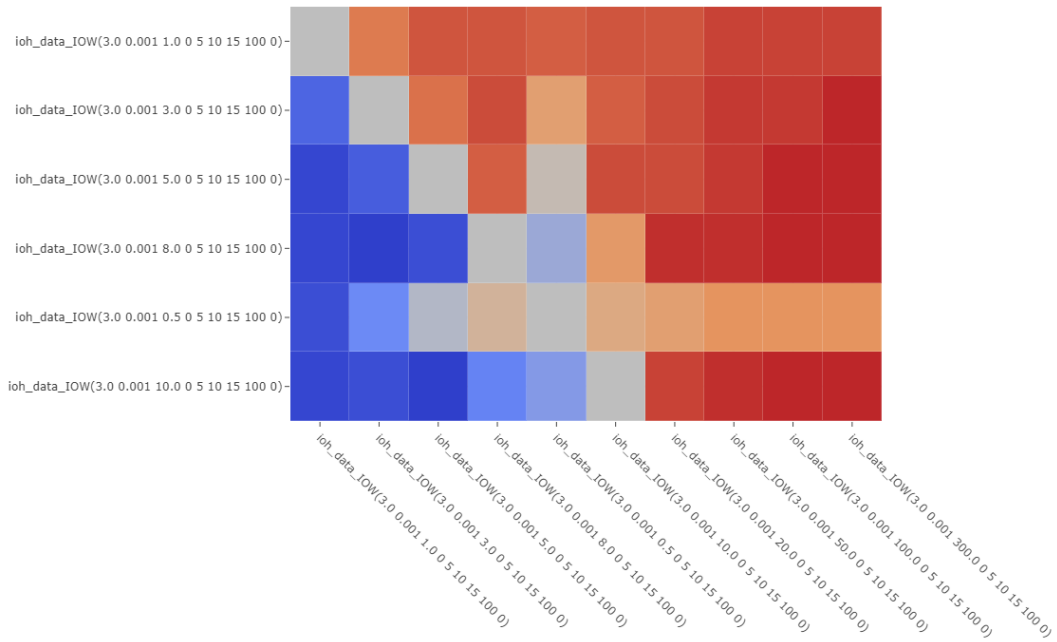

Figure 22: Heat map across all functions and dimensions for different values of the σ_{initial} (third parameter in brackets) of the IWO algorithm. Red means the algorithm on the y-axis outperforms the one on the x-axis.

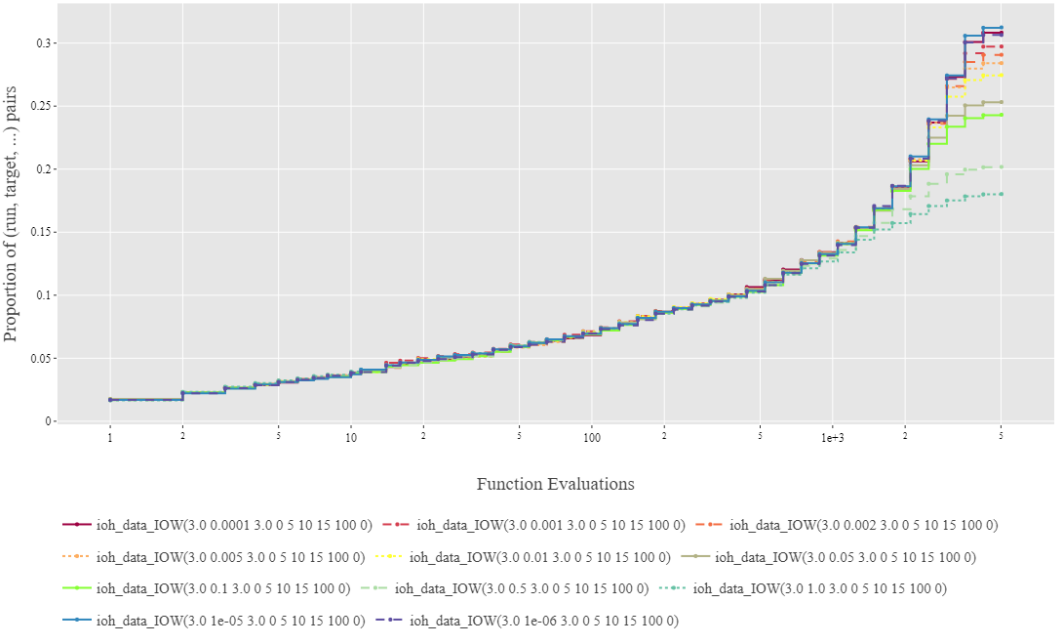

Figure 23: Aggregated ECDF across all functions and dimensions for different values of the σ_{final} (second parameter in brackets) of the IWO algorithm, using BBOB spacing for the automatically generated ECDF-targets.

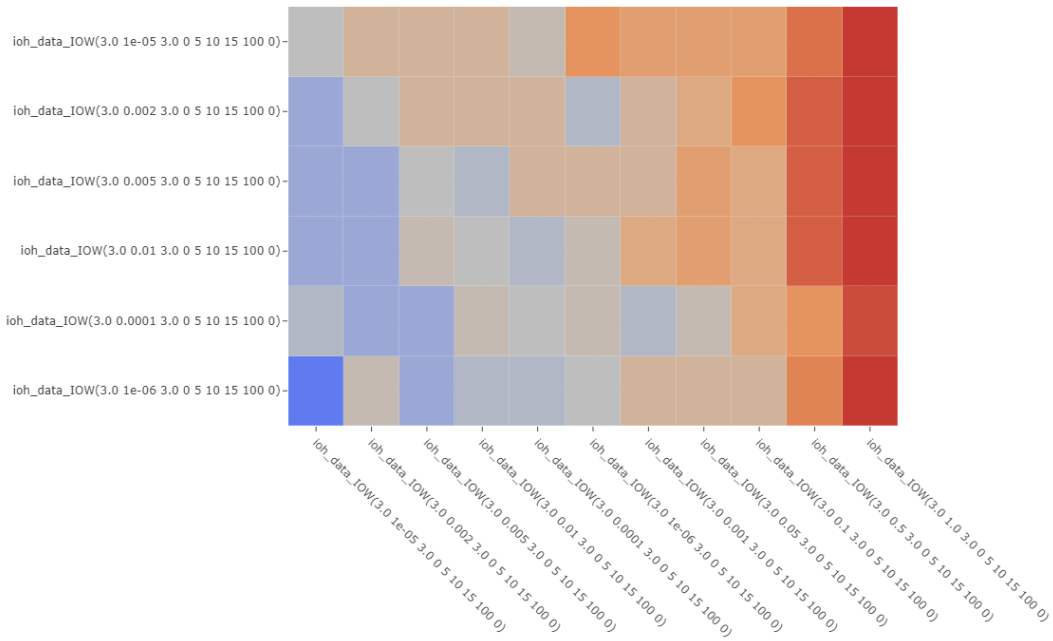

Figure 24: Heat map across all functions and dimensions for different values of the σ_{final} (second parameter in brackets) of the IWO algorithm. Red means the algorithm on the y-axis outperforms the one on the x-axis.

Symbol	**Description**	**Values**		
		Default values (Original)	**Tested values (Experiments)**	**Accepted range**
M_{max}	Maximum population size	15	10 – 500	10 – 30
S_{max}	Maximum number of seeds	5	1 – 10, 20, 50, 100	1 – 5
S_{min}	Minimum number of seeds	0	0 – 5	0
M_0	Initial population size	10	10	10
G	Total number of generations	100	5 – 500	∞
n	Nonlinear modulation index	3	0.1 – 50	2 – 8
σ_{initial}	Initial value of standard deviation	3.0	0.5 – 300.0	1 – 5
σ_{final}	Final value of standard deviation	0.001	1e-6 – 1.0	1e-5 – 0.01

Table 7: IWO parameter values used in the experiments.

4.6 Performance comparison

In this section, we benchmark the performance of the IWO algorithm against several well-established optimization techniques, including GA, a-CMA-ES, BIPOP-CMA-ES, and RS. The benchmarking process involves evaluating these algorithms on the 24 BBOB functions across different dimensions: 2, 5, 20, and 40.
The objective is to identify the scenarios in which IWO outperforms the other algorithms, thereby highlighting its strengths and potential advantages in various optimization contexts.
The parameter settings used for the IWO algorithm in this study are shown in Table 8. Where the number of generations is defined based on the problem dimension. Higher number of generations is used with higher dimensions to ensure a thorough exploration of the search space while maintaining a feasible computational budget for the exploitation process.

Dimensions	**Generations**	**Budget**
2	100	6000
5	500	30000
20	1000	50000
40	10000	500000

Symbol	**Description**	**Value**
M_0	Initial population size	10
M_{max}	Maximum population size	30
G	Total number of generations	100, 500, 1000 or 10000
d	Problem dimension	2, 5, 20 or 40
S_{max}	Maximum number of seeds	5
S_{min}	Minimum number of seeds	0
n	Nonlinear modulation index	8
$\sigma_{initial}$	Initial value of standard deviation	5
σ_{final}	Final value of standard deviation	1e-5

Table 8: IWO parameter values for performance comparison with GA, a-CMA-ES, BIPOP-CMA-ES and RS.

In the context of benchmarking optimization algorithms, the Empirical Cumulative Distribution Function (ECDF) is a crucial performance measure. The ECDF represents the proportion of problems for which an algorithm can achieve a given target function value within a specified number of function evaluations. In simpler terms, the ECDF provides a cumulative measure of an algorithm's success rate across different benchmark problems and targets, thus giving a comprehensive view of its overall performance.
As illustrated in Figures 29, 25, 30, 26, 31, 27, 35, and 28, the performance of various optimization algorithms exhibits notable trends across different problem dimensions. The IWO algorithm generally demonstrates competitive performance, particularly excelling in lower-dimensional problems where it outperforms both GA and RS, achieving faster convergence and lower fitness values, especially when finely tuned. However, as the dimensionality of the problems increases, IWO's relative performance declines; it consistently surpasses RS but does not outperform GA as markedly.

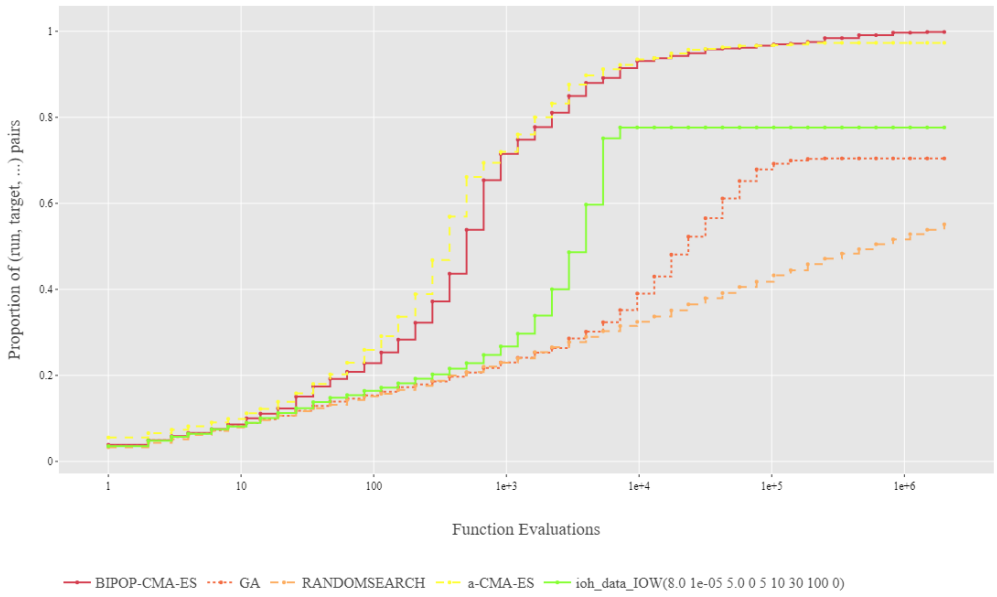

Figure 25: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 2. The figure shows the aggregated ECDF across all functions with a dimensionality of 2, using BBOB spacing for the automatically generated ECDF-targets. The y-axis tracks the loss, defined as target precision.

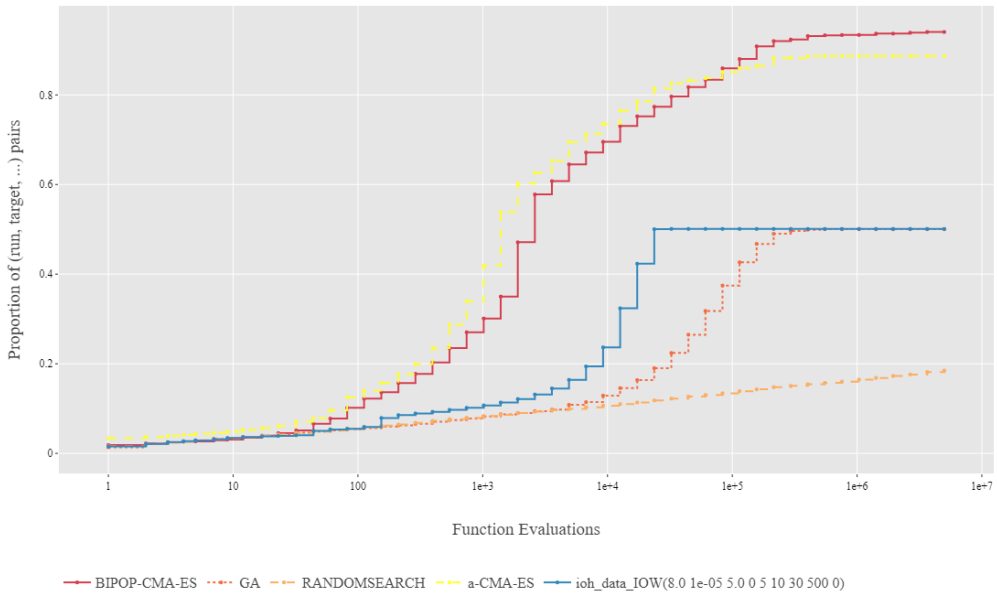

Figure 26: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 5. The figure shows the aggregated ECDF across all functions with a dimensionality of 5, using BBOB spacing for the automatically generated ECDF-targets. The y-axis tracks the loss, defined as target precision.

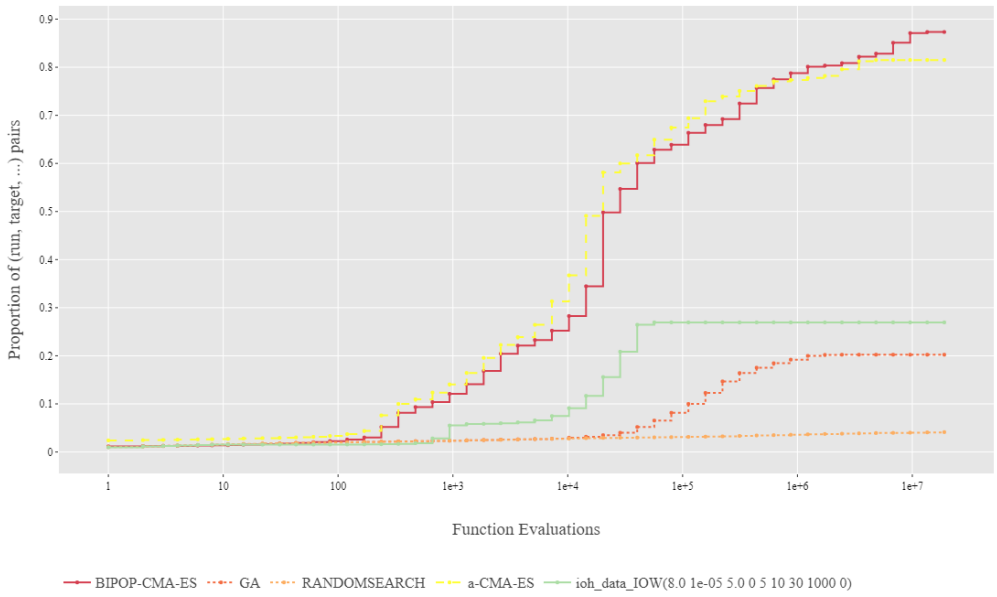

Figure 27: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 20. The figure shows the aggregated ECDF across all functions with a dimensionality of 20, using BBOB spacing for the automatically generated ECDF-targets. The y-axis tracks the loss, defined as target precision.

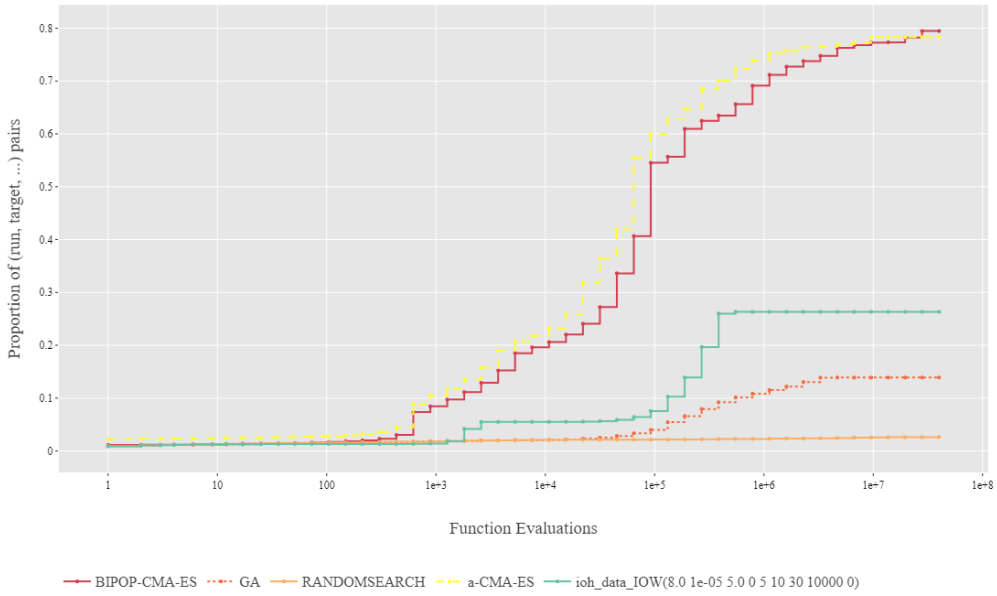

Figure 28: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 40. The figure shows the aggregated ECDF across all functions with a dimensionality of 40, using BBOB spacing for the automatically generated ECDF-targets. The y-axis tracks the loss, defined as target precision.

In contrast, the BIPOP-CMA-ES and a-CMA-ES algorithms consistently exhibit superior performance across most functions in all dimensions, showcasing their robustness and optimization capabilities. Notably, IWO achieves faster convergence and lower function values than BIPOP-CMA-ES and a-CMA-ES in specific functions. For example, in 2-dimensional problems, IWO excels in functions F3, F15, F22, and F23; in 5-dimensional problems, in functions F16 and F23; and even in higher-dimensional problems, such as in 20-dimensional problems, in functions F16, F20, F23, and F24 (see Appendix A for more details).

A noteworthy aspect of IWO's performance across all functions and dimensions is its strength during the exploitation phase. During the early stages, the algorithm is in the exploration phase, performing similarly to RS and GA. However, in the final stages, IWO effectively exploits the best-known solutions, significantly enhancing the search process in the surrounding area of these solutions. This characteristic is particularly advantageous in highly multi-modal functions with weak global structures, where high exploitation is needed to achieve a satisfactory fitness value, if not the optimal one, as observed in function F23 (see Figure 33). Additionally, IWO's exploitation capability is beneficial in problems with a global structure as in function F15 (see Figure 34).
The number of generations significantly impacts the algorithm's performance, often resulting in suboptimal outcomes when evaluated using anytime performance metrics. Therefore, it is recommended that the IWO algorithm be employed with parameters finely tuned according to a predetermined computational budget.

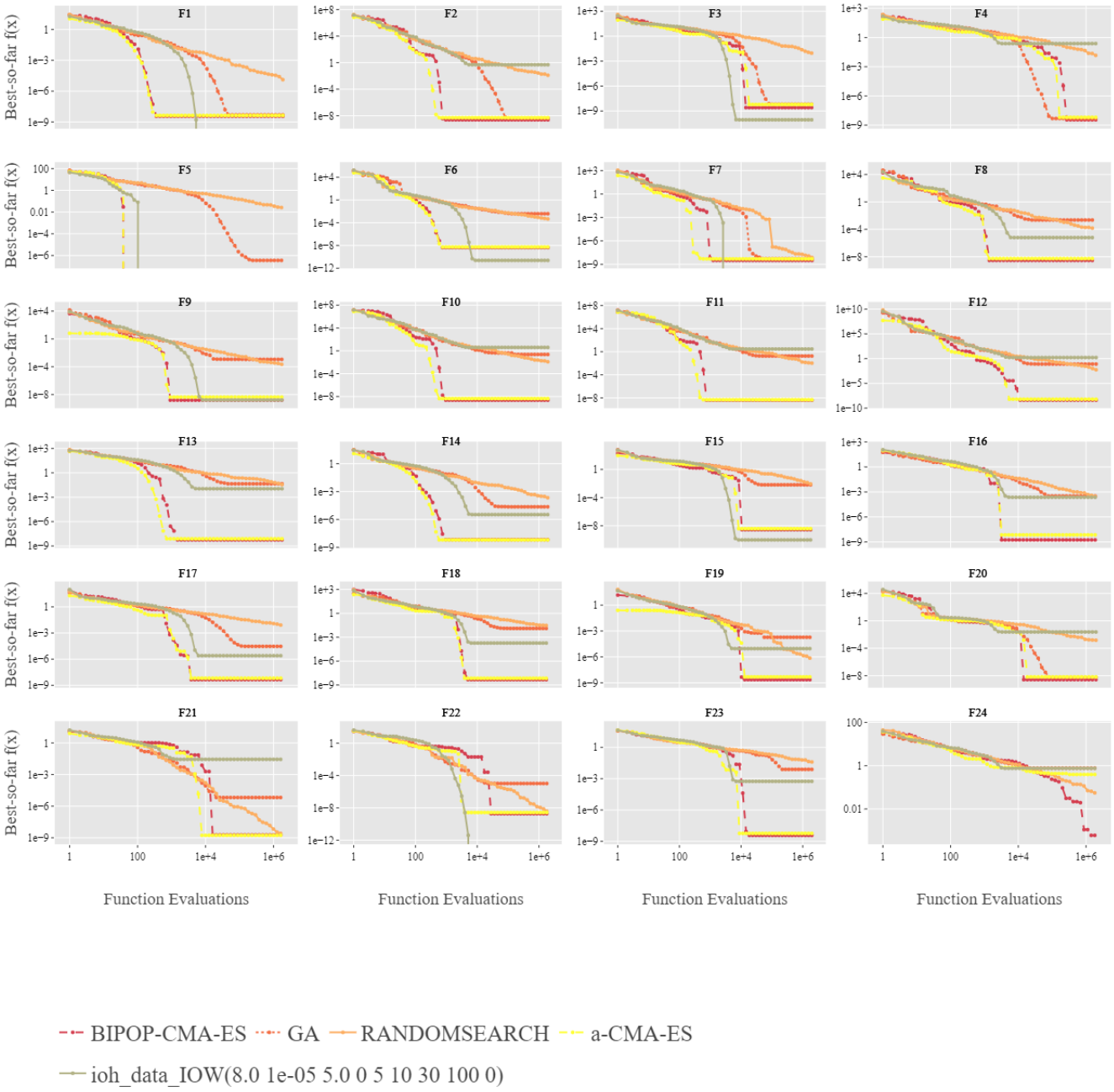

Figure 29: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 2. The figure shows "Best-so-far f(x)" versus function evaluations.

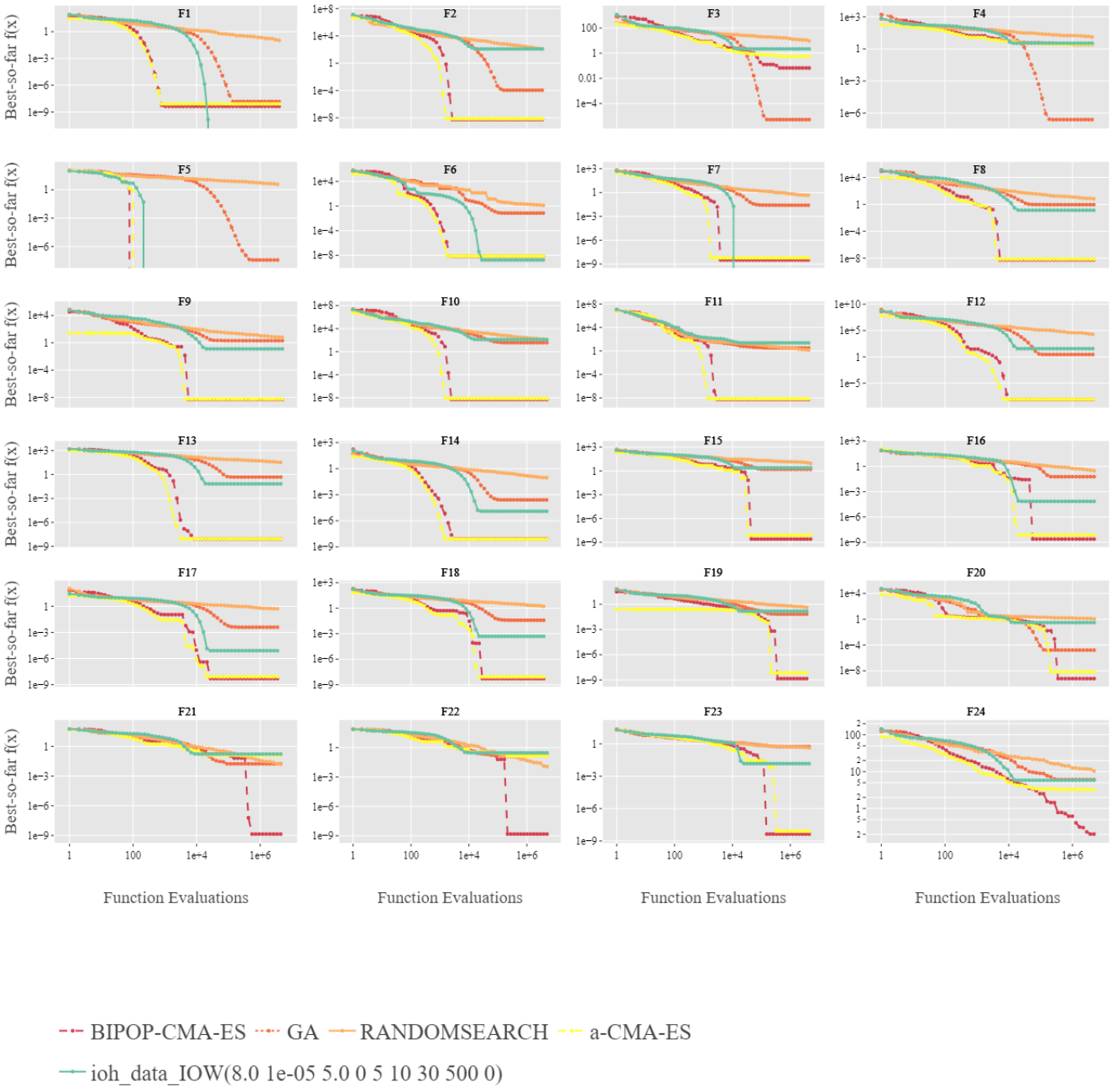

Figure 30: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 5. The figure shows "Best-so-far f(x)" versus function evaluations.

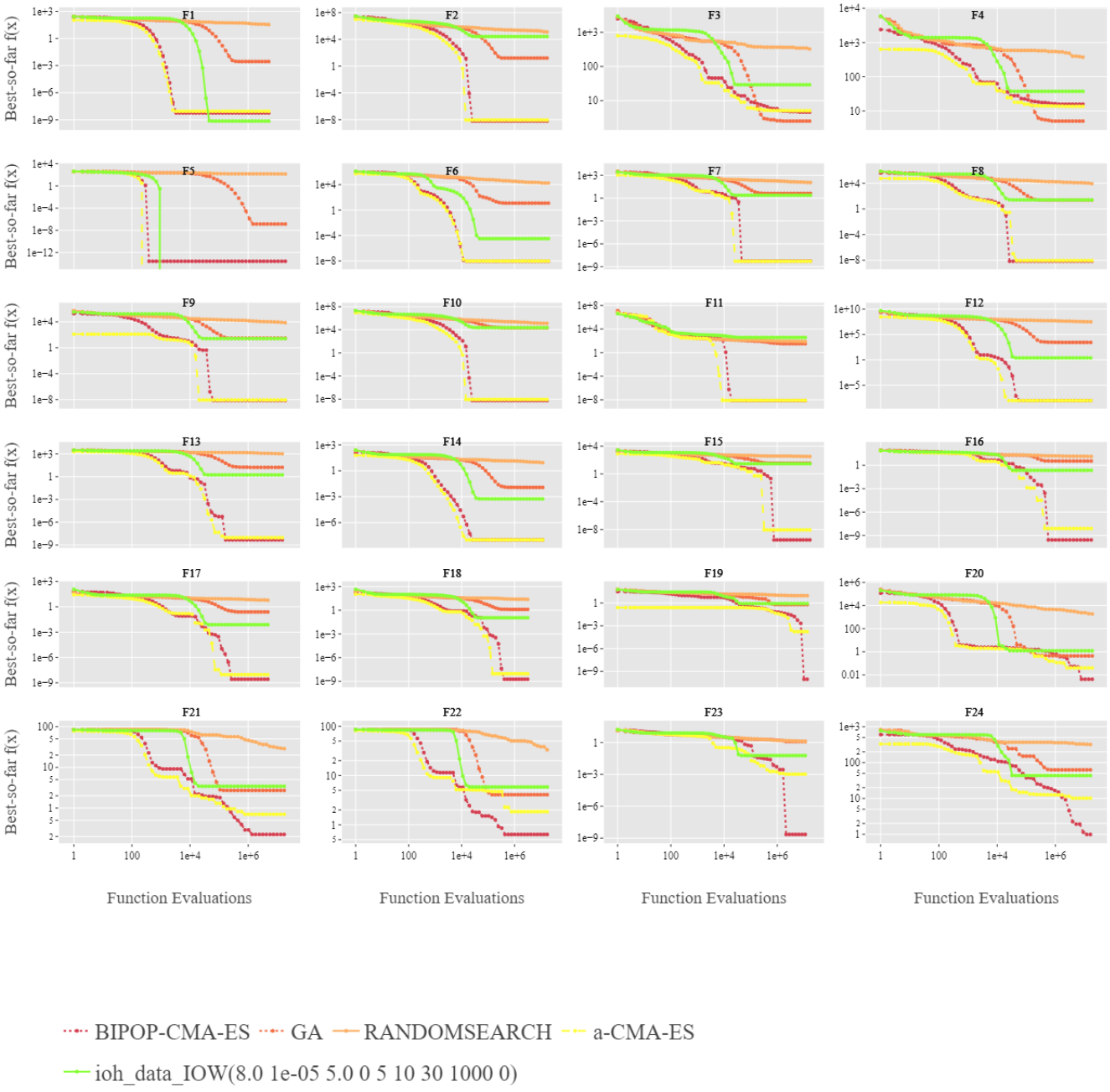

Figure 31: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 20. The figure shows "Best-so-far f(x)" versus function evaluations.

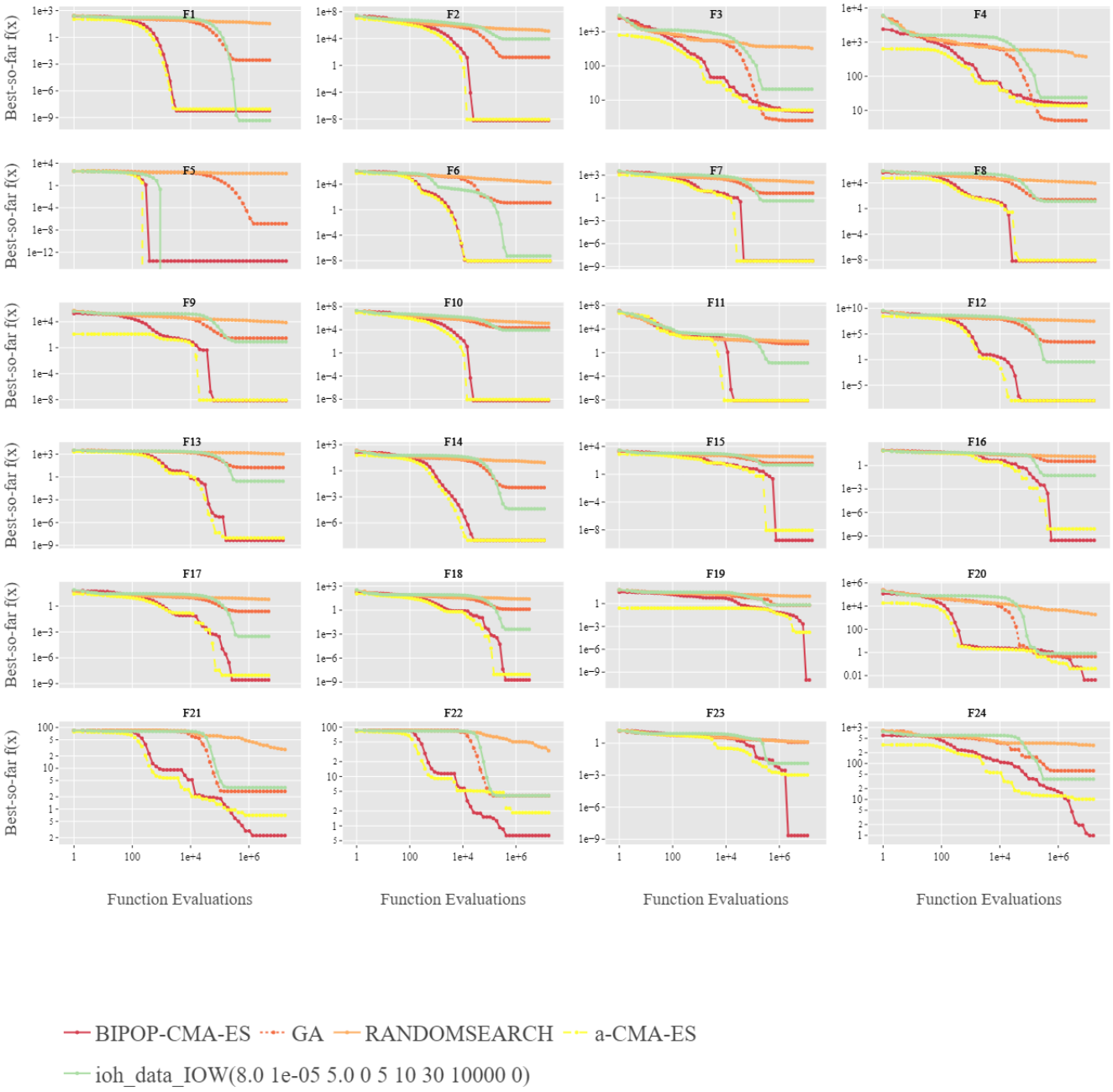

Figure 32: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across 24 functions (F1 to F24) with a dimensionality of 40. The figure shows "Best-so-far f(x)" versus function evaluations.

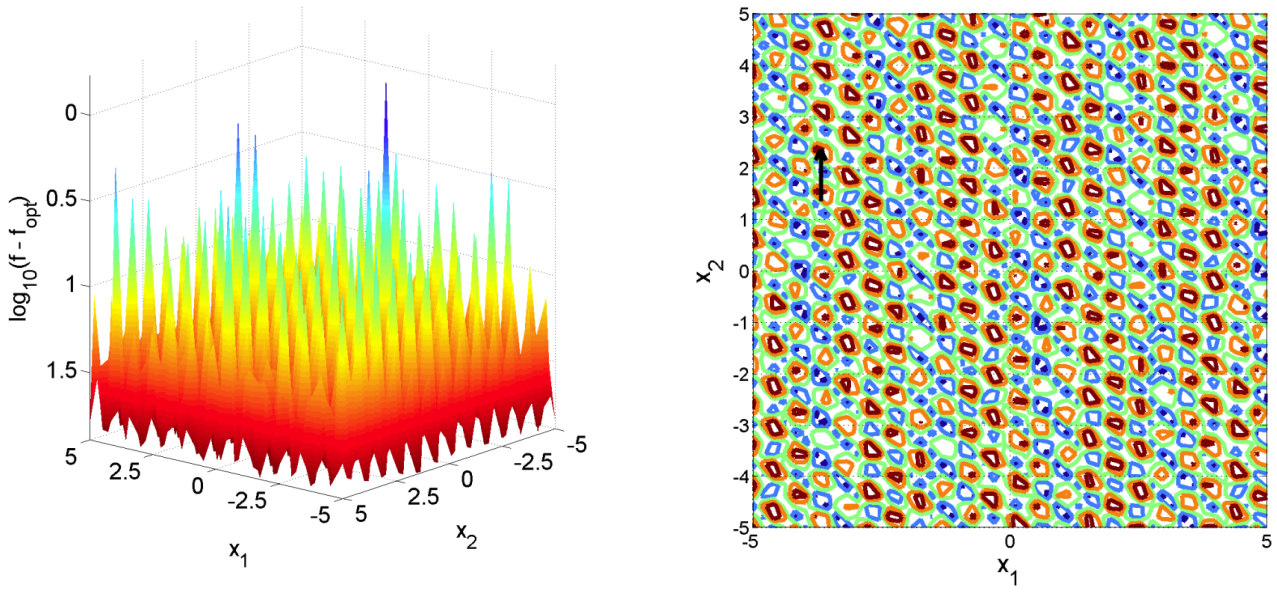

Figure 33: Surface plot (left) and 2D contour plot (right) of function F23 from the BBOB suite on the COCO platform [HAM+16], over 2-dimensional search space, illustrating the complex multi-modal landscape.

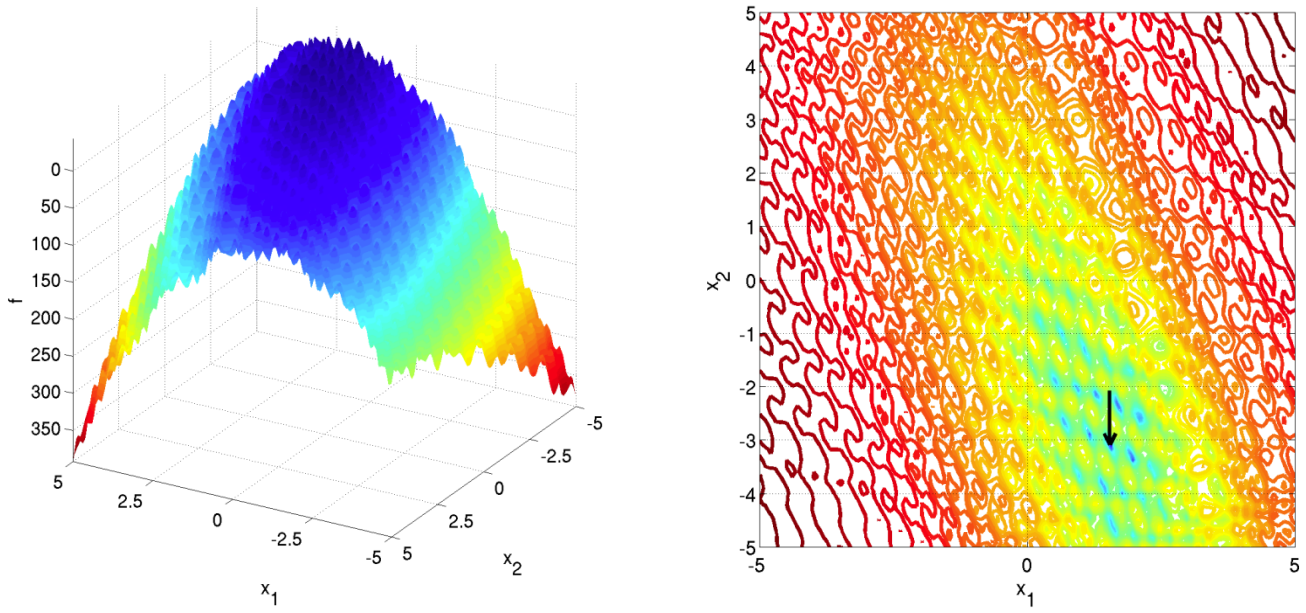

Figure 34: Surface plot (left) and 2D contour plot (right) of function F15 from the BBOB suite on the COCO platform [HAM+16], over 2-dimensional search space.

4.7 Experimental observations

Through these experiments, we validated the effectiveness of the IWO algorithm across various benchmark functions and compared its performance against other optimization algorithms. The following encapsulates the main observations, advantages, and disadvantages derived from this experimental study.

- IWO's strength is particularly notable during the exploitation phase, where it effectively enhances the search process around the best-known solutions.
- The performance of IWO tends to decline in higher-dimensional spaces compared to other optimization algorithms like BIPOP-CMA-ES and a-CMA-ES.
- The performance of IWO can be significantly improved by fine-tuning parameters such as the nonlinear modulation index, population size, and standard deviation values. Proper parameter tuning is crucial for achieving optimal performance.

5 Conclusions and Future work

In this thesis, we have explored the IWO algorithm, a nature-inspired optimization method modeled after the colonization behavior of weeds. Through a series of detailed experiments and analyses, we have validated the effectiveness of the IWO algorithm across various benchmark functions and compared its performance against other established optimization algorithms such as GA, a-CMA-ES, BIPOP-CMA-ES, and RS.
Performance-wise, the IWO performs well across a range of test functions, particularly in lower-dimensional problems. However, its performance varied significantly across different benchmark functions and dimensions. In comparison to the BIPOP-CMA-ES algorithm, IWO exhibited competitive performance in certain scenarios but generally lagged in high-dimensional spaces. The strengths of IWO were particularly notable during the exploitation phase of the optimization process.
The IWO algorithm demonstrated originality in its biological inspiration, distinguishing itself from other algorithms, such as GA, SA, and PSO. The concept of weed colonization, with its emphasis on seed dispersion and competitive exclusion, provided a unique approach to optimization.

Future research could concentrate on conducting experiments without the cutoff point used in the IOHprofiler, as this may yield significant findings and provide a more comprehensive evaluation of the algorithm's performance relative to other optimization algorithms. Moreover, an in-depth exploration of optimal parameter settings, designed to specific characteristics and dimensionality of various problems, could significantly enhance the algorithm's adaptability and effectiveness across a broader spectrum of optimization challenges. Additionally, investigating the relationship between parameter settings and the available computational budget is crucial, given that the performance of IWO is highly dependent on the budget. Establishing a mathematical relationship or model to optimize this interplay would be a valuable advancement in the practical application of IWO.

References

[CA21] Felipe Campelo and Claus Aranha. Evolutionary computation bestiary. https://github.com/fcampelo/EC-Bestiary, 2021. Accessed: 2024-06-12.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 26(1):1–13, 1996.

[DWY+18] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *CoRR*, abs/1810.05281, 2018.

[EK95] Russell Eberhart and James Kennedy. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, pages 1942–1948, 1995.

[HAM+16] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *CoRR*, abs/1603.08785, 2016.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[ML06] A.R. Mehrabian and C. Lucas. A novel numerical optimization algorithm inspired from weed colonization. *Ecological Informatics*, 1(4):355–366, 2006.

[Sör15] Kenneth Sörensen. Metaheuristics - the metaphor exposed. *ITOR*, 22(1):3–18, 2015.

[YK05] Ming Yuchi and Jong-Hwan Kim. Ecology-inspired evolutionary algorithm using feasibility-based grouping for constrained optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1455–1461, Edinburgh, UK, September 2005.

A Appendix

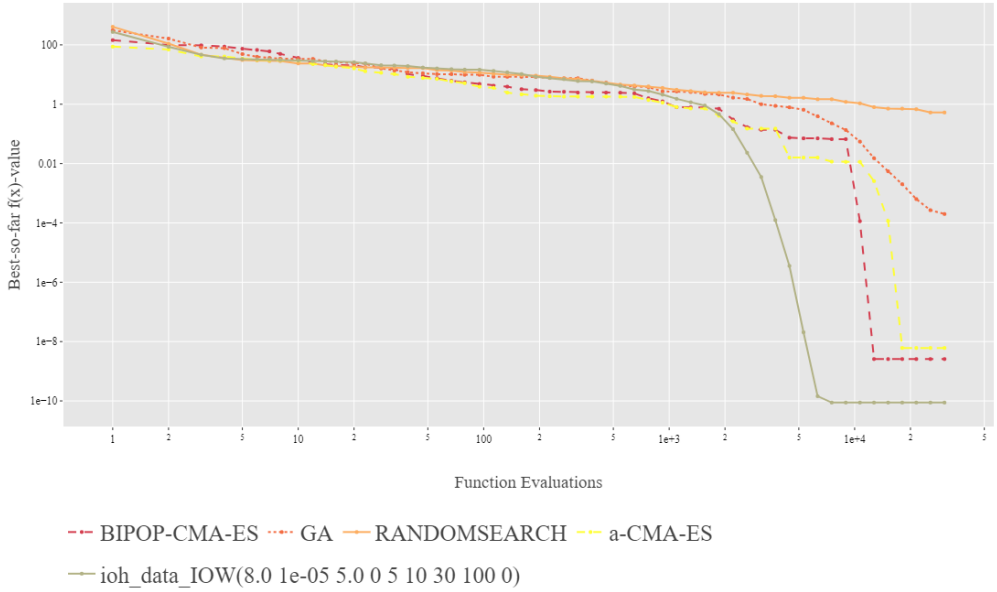

Figure 35: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F3 with a dimensionality of 2. The figure shows "Best-so-far f(x)" versus function evaluations.

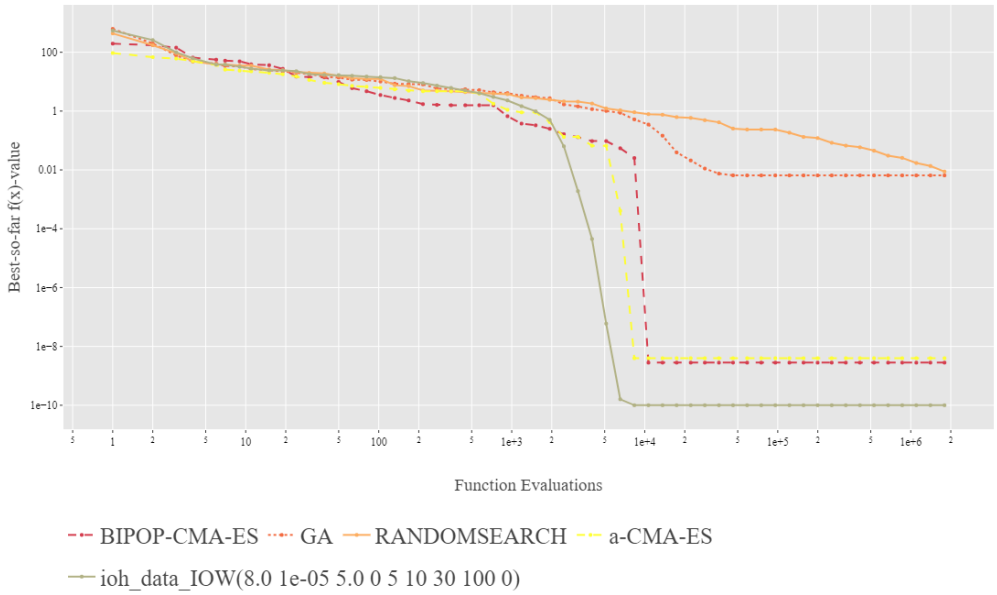

Figure 36: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F15 with a dimensionality of 2. The figure shows "Best-so-far f(x)" versus function evaluations.

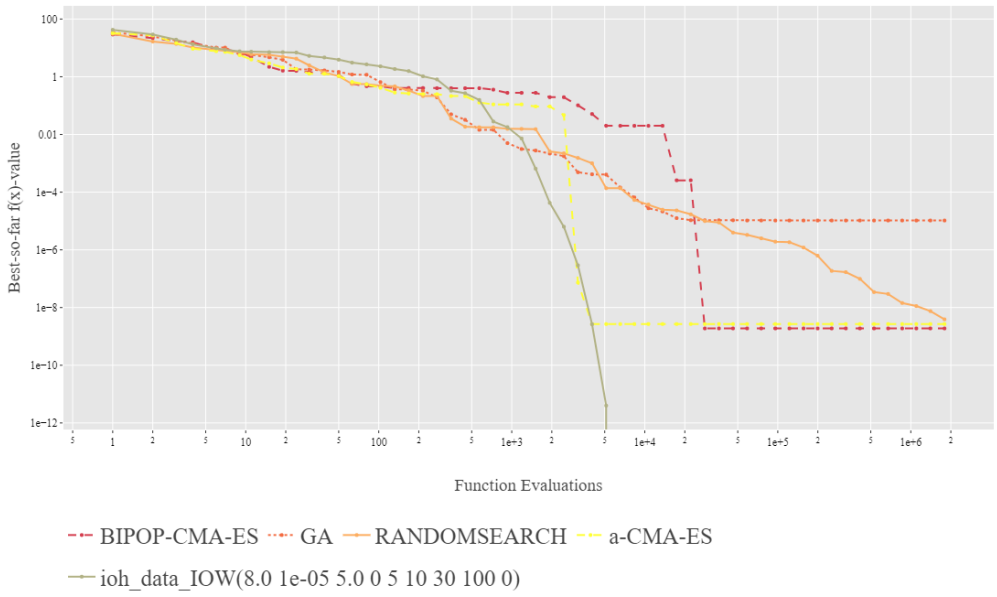

Figure 37: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F22 with a dimensionality of 2. The figure shows "Best-so-far f(x)" versus function evaluations.

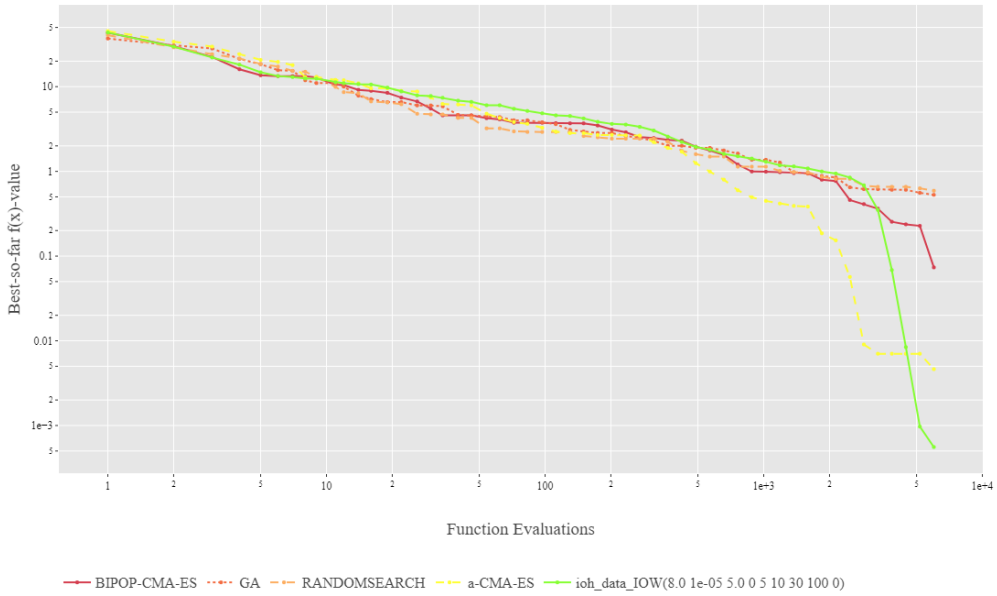

Figure 38: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F23 with a dimensionality of 2. The figure shows "Best-so-far f(x)" versus function evaluations.

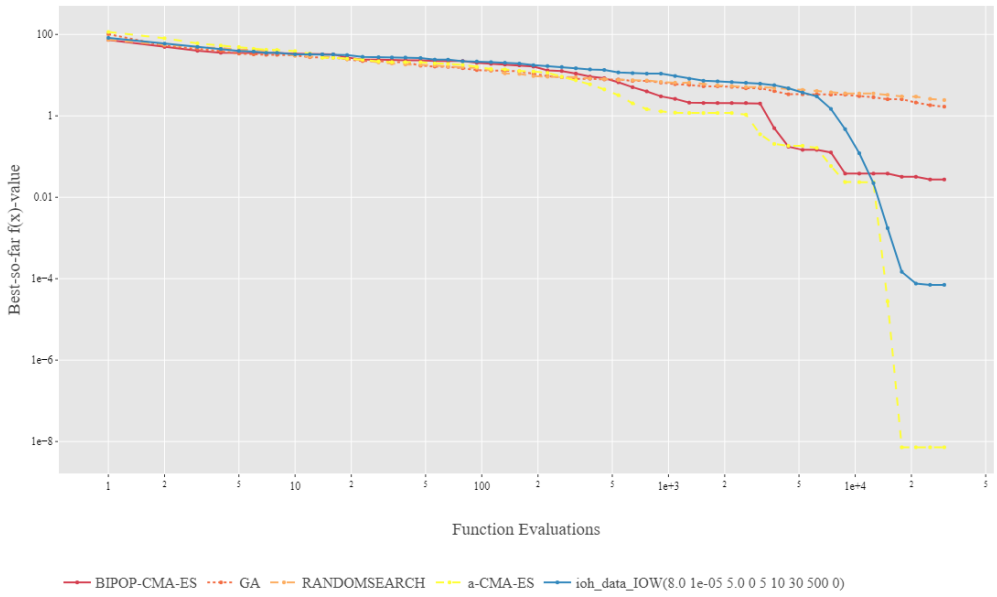

Figure 39: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F16 with a dimensionality of 5. The figure shows "Best-so-far f(x)" versus function evaluations.

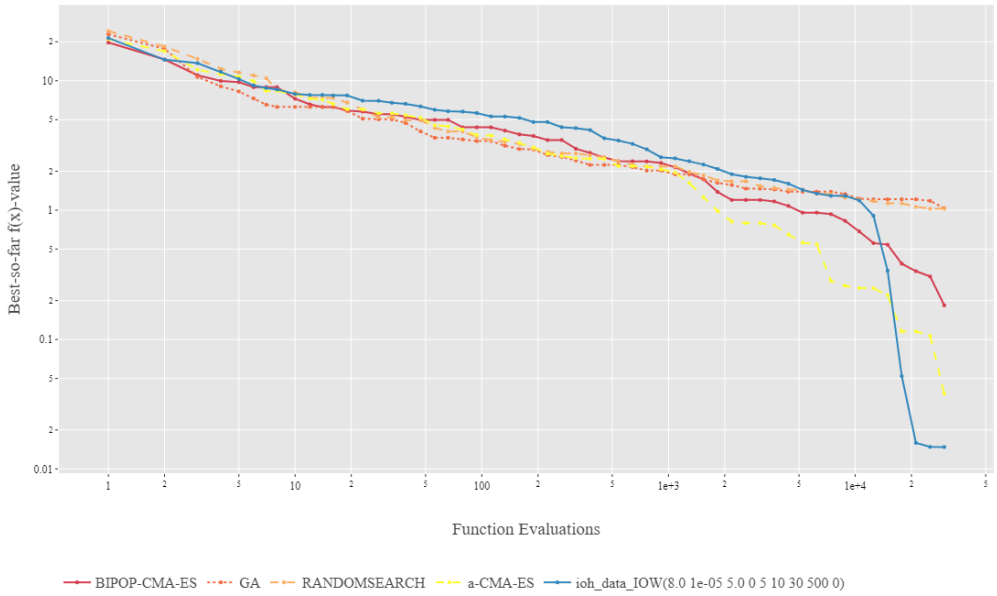

Figure 40: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F23 with a dimensionality of 5. The figure shows "Best-so-far f(x)" versus function evaluations.

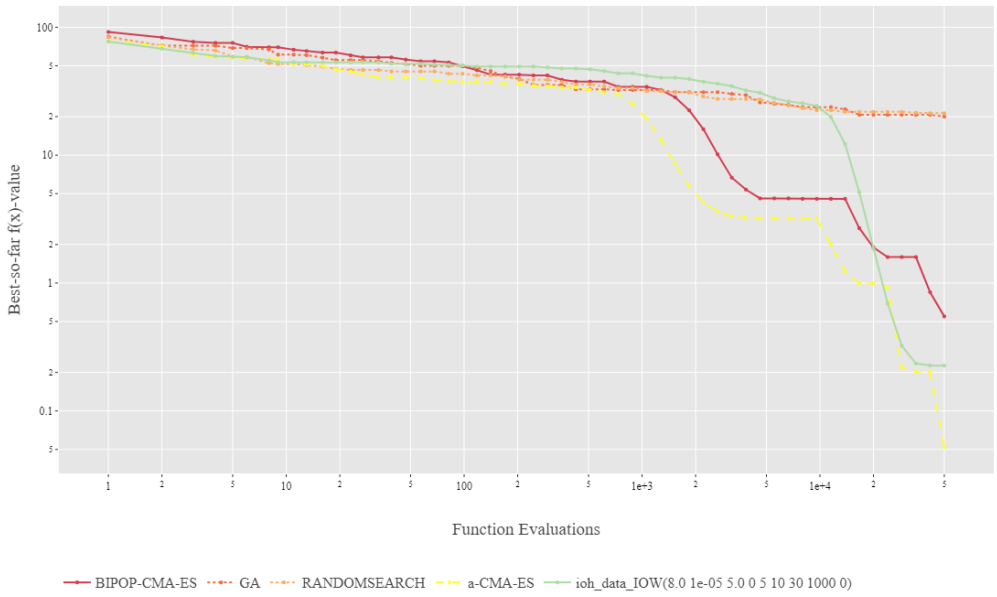

Figure 41: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F16 with a dimensionality of 20. The figure shows "Best-so-far f(x)" versus function evaluations.

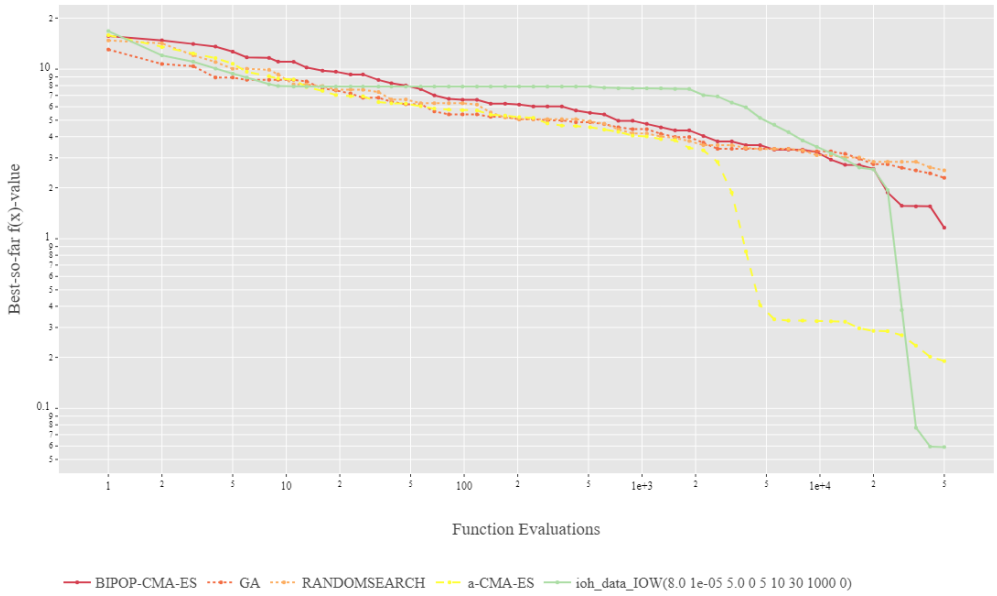

Figure 42: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F23 with a dimensionality of 20. The figure shows "Best-so-far f(x)" versus function evaluations.

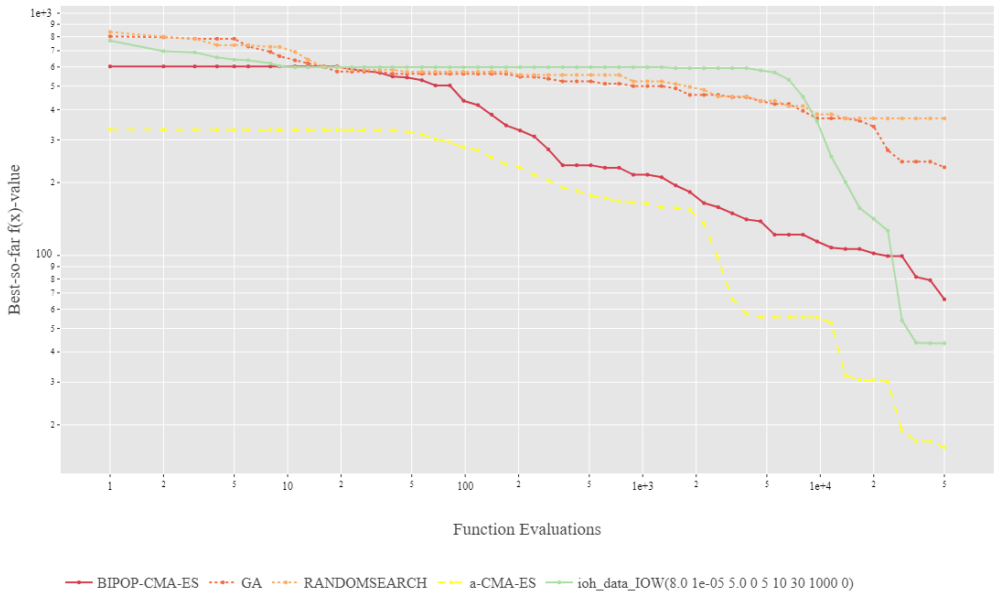

Figure 43: Comparison of the performance of BIPOP-CMA-ES, GA, RS, a-CMA-ES, and IWO across on F24 with a dimensionality of 20. The figure shows "Best-so-far f(x)" versus function evaluations.

A.1 IWO implementation in Python

The complete implementation is also available on GitHub at the following link: https://github.com/BarmadShreef4/iwo-project.git.

```
import ioh
import sys
import numpy as np


########################################## IWO ##########################################
class IWO:
    """Invasive Weed Optimization algorithm

    Args:
        min_seeds: minimum number of seeds allowed per individual
        max_seeds: maximum number of seeds allowed per individual
        n: the nonlinear modulation index
        final_sigma: final value of sigma (standard deviation)
        init_sigma: initial value of sigma (standard deviation)
        curr_sigma: current value of sigma
        init_pop_size: initial population size
        max_pop_size: maximum population size
        num_generations: total number of generations
    """

    def __init__(
        self,
        min_seeds: int,
        max_seeds: int,
        n: float,
        final_sigma: float,
        init_sigma: float,
        init_pop_size: int,
        max_pop_size: int,
        num_generations: int,
        save_visu_data: int = 0,
    ):
        self.min_seeds: int = min_seeds
        self.max_seeds: int = max_seeds
        self.n: float = n
        self.final_sigma: float = final_sigma
        self.init_sigma: float = init_sigma
        self.curr_sigma: float = 0.0
        self.init_pop_size: int = init_pop_size
        self.max_pop_size: int = max_pop_size
```

```
        self.num_generations: int = num_generations
        self.save_visu_data: int = save_visu_data
        self.population = []
        self.fitness = []

    def _save_visu_data(self, generation):
        """Save the positions and fitness of the population members to a file.
           For visualization purposes only.

        Args:
            generation (int): The actual generation number.
        """
        for agent_index in range(len(self.population)):
            # Append the data for each member of the population along
            # with the generation number
            output_file.write(
                ",".join(
                    map(
                        str,
                        [generation]
                        + np.append(
                            self.population[agent_index],
                            self.fitness[agent_index],
                        ).tolist(),
                    )
                )
                + "\n"
            )

    def _calculate_sigma(self, curr_generation: int) -> None:
        """Calculates the Spatial Dispersal coefficient (eq. 1).

        Args:
            curr_generation: Current generation number.
        """
        coef = (
            (self.num_generations - curr_generation) / (self.num_generations)
        ) ** self.n

        self.curr_sigma = (
            coef * (self.init_sigma - self.final_sigma) + self.final_sigma
        )

    def _initialize_population(self, problem):
        self.population = np.random.uniform(
```

```
            low=problem.bounds.lb,
            high=problem.bounds.ub,
            size=(self.init_pop_size, problem.meta_data.n_variables),
        )
        # Evaluate fitness for each agent (weed)
        self.fitness = problem(self.population)

    def _reproduction_phase(self):
        # Sort agents based on fitness
        sorted_indices = np.argsort(self.fitness)
        self.population = self.population[sorted_indices]
        self.fitness.sort()

        # Calculate the number of seeds for each agent (weed) based
        # on its rank (its fitness)
        n_seeds = np.round(
            np.interp(
                self.fitness,
                [self.fitness[0], self.fitness[-1]],
                [self.max_seeds, self.min_seeds],
            )
        ).astype(int)
        return n_seeds

    def _spatial_dispersal_phase(self, problem, generation, n_seeds):
        # calculat the sigma (standard deviation) for spatial randomness
        # for the current iteration (generation)
        self._calculate_sigma(generation)

        # iterate through each agent in the population
        for agent_index in range(len(self.population)):
            # Generate positions for the seeds based on normal distribution
            # with the standard deviation curr_sigma
            seeds = np.random.normal(
                self.population[agent_index],
                self.curr_sigma,
                size=(n_seeds[agent_index], len(self.population[agent_index])),
            )

            if seeds.size != 0:
                # Evaluate fitness for each seed
                seeds_fitness = problem(seeds)

                # update population
                self.fitness = np.concatenate((self.fitness, seeds_fitness))
```

```
            self.population = np.concatenate((self.population, seeds))

    def _competitive_exclusion_phase(self):
        # Sort agents based on fitness
        sorted_indices = np.argsort(self.fitness)
        self.population = self.population[sorted_indices]
        self.fitness.sort()

        # eliminate agents with lower fitness to reach the maximum
        # allowable population size
        if len(self.population) > self.max_pop_size:
            self.population = self.population[: self.max_pop_size]
            self.fitness = self.fitness[: self.max_pop_size]

    def __call__(self, problem: ioh.problem.RealSingleObjective) -> None:
        "Optimize the problem with the Invasive Weed Optimization algorithm"

        ################### Initialization phase ###################
        self._initialize_population(problem)

        # iterating through each generation
        for generation in range(self.num_generations):
            ######################## Reproduction phase ########################
            n_seeds = self._reproduction_phase()

            if self.save_visu_data:
                # save population progress data across generations
                # (for visualization)
                self._save_visu_data(generation)

            ##################### Spatial dispersal phase #####################
            self._spatial_dispersal_phase(problem, generation, n_seeds)

            ################### Competitive exclusion phase ###################
            self._competitive_exclusion_phase()


if __name__ == "__main__":
    # Retrieve command-line arguments
    args = sys.argv[1:]
    usage_info = [
        "\nUsage: python3 my_IWO.py [n] [final_sigma] [init_sigma]",
        "[min_seeds] [max_seeds] [init_pop_size] [max_pop_size]",
        "[num_generations] [save_visu_data] [output_directory]\n\n",
        "-----------------n: the nonlinear modulation index\n",
```

```
    "       final_sigma: final value of sigma (standard deviation)\n",
    "        init_sigma: initial value of sigma (standard deviation)\n",
    "         min_seeds: minimum number of seeds allowed per individual\n",
    "         max_seeds: maximum number of seeds allowed per individual\n",
    "     init_pop_size: initial population size\n",
    "     max_pop_size: maximum population size\n",
    "   num_generations: total number of generations\n",
    "   save_visu_data: whether to save data for visualisation or not\n",
    "  output_directory: directory where to store output files."
    "\n\nEx: python3 my_IWO.py 3.0 0.1 10.0 1 5 5 40 50 1 ./results/\n",
]

# Check if the correct number of arguments is provided
if len(args) != 10:
    print(*usage_info)
    sys.exit(1)

# Convert arguments to appropriate types
n, final_sigma, init_sigma = map(float, args[:3])
(
    min_seeds,
    max_seeds,
    init_pop_size,
    max_pop_size,
    num_generations,
    save_visu_data,
) = map(int, args[3:-1])
output_directory = args[-1]

# Create a folder name for this run based on used parameters
ioh_data_folder_name = (
    "ioh_data_IOW("
    + str(n)
    + "_"
    + str(final_sigma)
    + "_"
    + str(init_sigma)
    + "_"
    + str(min_seeds)
    + "_"
    + str(max_seeds)
    + "_"
    + str(init_pop_size)
    + "_"
    + str(max_pop_size)
```

```
    + "-"
    + str(num_generations)
    + "-"
    + str(save_visu_data)
    + ")"
)

if save_visu_data == 1:
    # open the a file for writing population progress data
    # across generations
    output_file = open("population_progress.csv", "w")

experiment = ioh.Experiment(
    algorithm=IWO(
        min_seeds,
        max_seeds,
        n,
        final_sigma,
        init_sigma,
        init_pop_size,
        max_pop_size,
        num_generations,
        save_visu_data,
    ),  # An algorithm instance
    fids=list(range(1, 25)),  # the id's of the problems we want to test
    iids=list(range(1, 6)),  # the instances
    dims=[
        2,
        5,
        20,
        40,
    ],  # the dimensions (number of variables for each agent)
    reps=5,  # the number of runs for each combination (fids, iids, dims),
    problem_class=ioh.ProblemClass.BBOB,
    algorithm_name=ioh_data_folder_name,
    remove_data=True,  # delete data in the folder (Not the .zip file)
    zip_output=True,
    algorithm_info="",
    store_positions=True,
    merge_output=True,
    folder_name=ioh_data_folder_name,
    output_directory=output_directory,
)

print("started:", ioh_data_folder_name)
```

```
experiment()

if save_visu_data == 1:
    # close the file
    output_file.close()
```