Universiteit Leiden
The Netherlands

Opleiding Informatica

Illuminating the EC-bestiary: Conceptual Analysis and Benchmarking of the Coyote Optimization Algorithm

Ivar van der Spoel

Supervisor: Dr Anna V. Kononova, MSc Haoran Yin

BACHELOR THESIS

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

18/06/2024

Abstract

The EC-Bestiary is a comprehensive collection of bio-inspired optimization algorithms, yet it lacks thorough benchmarking studies. This thesis evaluates the performance of the Coyote Optimization Algorithm (COA) in comparison to other well-established heuristics. We use standardized benchmarking tools such as IOHprofiler and the BBOB suite to analyze COA's strength in solving various optimization problems. Our extensive experiments reveal that COA exhibits competitive performance, particularly in multi-modal and weak structure functions, and exceptional results in low-dimensional function evaluations while offering unique contributions to the EC-Bestiary. Additionally, we provide a detailed statistical analysis and comparisons with algorithms like PSO, ABC, and CMA-ES. Our findings suggest that COA's social behavior-inspired mechanisms can enhance optimization outcomes in specific scenarios. This work aims to guide future research and application of bio-inspired optimization algorithms in diverse problem domains, contributing valuable data to the growing field of evolutionary computation.

Contents

1 Introduction

In this section the current situation is discussed, and why this leads to the research question. Lastly, a brief description of all the chapters is given.

1.1 The situation

The EC-Bestiary comprises a collection of bio-inspired optimization techniques that draw inspiration from nature [Cam24]. Over time, it has become home to numerous newly-developed evolutionary algorithms, whose inner working is inspired by nature. However, despite this rich diversity, many studies criticize an overwhelming number of these bio-inspired algorithms, because many of these algorithms are not tested rigorously enough. It remains unclear about the actual potential of individual algorithms, the shared characteristics among them, on what they strongly differ, and how the optimization of various problems can effectively benefit from their unique features[CA23].

The No Free Lunch Theorem [AAPV19] states that it is impossible for a singular algorithm to perform the best over all set of problems. This means that there always will be another set of problems it does not perform optimally. It is important to know which algorithm to choose with a certain problem. To know which algorithm outperforms others, we will benchmark them using the most established benchmarking principles [BBDB+20].

1.2 Thesis overview

In this thesis, we will be looking at one of these bio-inspired optimization algorithms in particular: the Coyote Optimization Algorithm (COA). Our main research question is:

How does the Coyote Optimization Algorithm perform compared to other established heuristics in solving optimization problems, and what unique contributions can it make to the EC-Bestiary collection?

To answer this question, we need to gather information about this topic and all it's related work, which will be done in Section 2. This provides a comprehensive review of existing literature on COA and other bio-inspired optimization algorithms. It also discusses the benchmarking principles and tools like IOHprofiler and the BBOB suite used in the analysis. After that the relevant definitions are mentioned in Section 3. Then in Section 4 we will discuss the method of how we want to answer the research question. This includes a detailed explanation of the methodologies employed to benchmark COA against other optimization algorithms. This includes the description of tools like IOHprofiler for performance analysis and Matplotlib for external analysis. Most importantly, the algorithm's description is given with certain tweaks and testing the correctness of the implementation in Section 5. Then we will discuss the findings obtained from benchmarking COA on the BBOB problems in Section 6, including statistical analysis and comparisons with other established heuristics. Lastly, we will conclude and discuss the project in Section 7 with a final remarks on the significance of COA in the field of bio-inspired optimization.

2 Related Work

It is a necessity to read up on related work regarding this topic as there is an abundance of information available, and there is a need to clarify which is relevant.

Firstly, the Coyote Optimization Algorithm (COA) is a niche metaheuristic inspired by the social behavior of coyotes. The original publication on COA writes about the algorithm and mathematic details, showcasing the strength in organization and behaviors observed in coyote packs compared to other algorithms which do not use this feat, and later on it becomes obvious as to why this is a positive attribute. The algorithm's performance was initially benchmarked against various other optimization algorithms such as Particle Swarm Optimization (PSO)[WTL18], Artificial Bee Colony (ABC)[Kar10], Symbiotic Organisms Search (SOS)[CP14], Grey Wolf Optimization (GWO)[MML14], Bat-inspired Algorithm (BA)[Yan10], and Firefly Algorithm (FA) [Yan09]. Very briefly, the original publication goes in-depth on a specific, considered to be, difficult optimization problem with all the before mentioned algorithms. The details of the benchmarking will be discussed in Section 6.3. [PC18, OK97].

When we will look at the performance of COA in detail, we will be using IOHprofiler. the IOHprofiler platform is an invaluable tool developed by LIACS for the benchmarking and analysis of optimization heuristics. It offers a comprehensive suite of tools for evaluating, comparing, and visualizing the performance of optimization algorithms. IOHprofiler's integration of the BBOB (Black-Box Optimization Benchmarking) suite enables the rigorous testing of continuous optimizers on a variety of standardized benchmark problems, such as the BBOB problems [DWY+18].

Just like IOHprofiler using the BBOB suite, many other numerous studies have used the BBOB suite for performance evaluation of optimization algorithms. The BBOB benchmarks are a set of standardized test functions that are widely recognized in the optimization community. They provide a robust framework for assessing various aspects of an algorithm's performance, including its ability to cover separable, moderately conditioned, ill-conditioned, multi-modal, and weak structure functions [NHB21, HAR+21].

Not only is the IOHprofiler suite a source of algorithm implementation and performance, but the Opytimizer library offers implementations of various optimization algorithms, including COA [GHdRP19]. However, for more detailed analyses, tools such as Matplotlib are employed. Matplotlib is a commonly used Python library that enables the creation of graphs to lead to deeper insights into algorithm performance and behavior that are not achievable with IOHprofiler alone [Hun07].

While using all these algorithms and benchmarking tools, the principles of benchmarking provide a structured approach to evaluating optimization algorithms. These principles emphasize the importance of comprehensive and systematic performance evaluations to determine the strengths and weaknesses of the algorithms in question [BBDB+20].

LIACS has conducted it's own large analysis on numerous meta heuristics, including COA, with an implementation of COA using Opytimizer, and if things are done properly, we expect to find similar results as to what the large-scale benchmarking study found [VDW+24].

While IOHprofiler provides a robust platform for performance analysis, certain implementations, such as those from Opytimizer, are not directly compatible. This means using external analyses tools like Matplotlib is necessary to check qualities of different implementations such as correctness.

The related work highlights the importance of using a combination of standardized benchmarking suites like BBOB, powerful analysis tools like IOHprofiler, and custom visualization libraries such as Matplotlib to comprehensively evaluate and understand the performance of optimization algorithms like COA. This approach ensures that the strengths and weaknesses of COA are thoroughly examined, providing insights for its application to various optimization problems.

3 Definitions

In this thesis, certain terminology is used frequently. This section elaborates on some of these key definitions, which appear multiple times throughout the thesis.

Meta-heuristic problems Meta-heuristic problems involve solving optimization problems using meta-heuristic algorithms, which are high-level problem-independent algorithms, an example set of these problems are the BBOB problems [OK97].

Swarm-based intelligence In the context of meta-heuristic algorithms, swarm-based intelligence refers to a type of algorithm which has a self-organized systems, made up of agents that interact locally with one another and with their environment [CK17].

BBOB BBOB (Black-Box Optimization Benchmarking) refers to a suite of benchmark functions used for evaluating the performance of optimization algorithms. These functions are designed to test the limits of algorithms under various conditions [HAR+21].

Dimensions In optimization, dimensions refer to the number of variables or parameters that the optimization algorithm needs to adjust in order to find the optimal solution. A certain dimensionality refers to how many dimensions of space there are.

Exploit and explore The exploit-explore dilemma in optimization refers to the trade-off between exploiting known good solutions to find even better ones in the same region in the search space or exploring new solutions to discover potentially completely different, and better areas in the search space.

Objective function An objective function is a mathematical function that the optimization algorithm seeks to minimize or maximize. It quantifies the quality of a solution in terms of its fitness or budget.

IOH IOH (Interactive Online Benchmarking) is a framework for benchmarking optimization heuristics. It provides tools for the evaluation, comparison, and visualization of the performance of optimization algorithms[DWY+18].

a-CMA-ES and BIPOP-CMA-ES a-CMA-ES (adaptive Covariance Matrix Adaptation Evolution Strategy) and BIPOP-CMA-ES are variants of the CMA-ES algorithm, which is a stochastic, derivative-free method for continuous optimization. This algorithm is used as the baseline benchmark, as it is consistently well performing across all BBOB functions and dimensions[Han06].

Random Search The most basic form of a meta-heuristic algorithm. This algorithm tries to find an optimal solution by trying random solutions in the search space [HI24].

Sphere function The Sphere function is a common test function used in optimization. It is defined in an n-dimensional space as:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$$

It is a symmetric paraboloid with a convex shape and no local minima, and one global optimum.

F21 F21 refers to a specific function within the BBOB benchmark suite, namely the Gallagher's Gaussian 101-me Peaks Function. Each function in this suite is designed to test different aspects of optimization algorithms. This function in particular is considered to be the most challenging, as it has the most peaks and is considered most sporadic [HAR+21].

COA COA (Coyote Optimization Algorithm) is a meta-heuristic algorithm inspired by the social behavior of coyotes, which we intend to benchmark and analyse [PC18].

Packs in COA COA has packs of coyotes. Each coyote is assigned to a pack, and the packs are a group of subsets of the population.

opytimizer Opytimizer is a Python library for building and testing optimization algorithms. It provides implementations of various optimization techniques, including COA [GHdRP19].

ECDF ECDF (Empirical Cumulative Distribution Function) is a statistical tool used to describe the distribution of data points. It is used in optimization to visualize the performance of algorithms across multiple runs[ioh24].

4 Method

It is important to establish the method used to answer the research question. In this thesis, there will be multiple ways to determine this. Namely, IOHprofiler to perform data analysis, alongside matplotlib with self-made scripts. Moreover, COA has multiple implementations, including the original one available on GitHub[Jkp] and the implementation from Opytimizer.

4.1 IOH and BBOB

The IOHprofiler tool will be crucial to help understand COA. IOHprofiler includes benchmarking problems such as the widely used BBOB problems, and has established meta-heuristic algorithms benchmarked and available to compare to.

4.1.1 IOH description

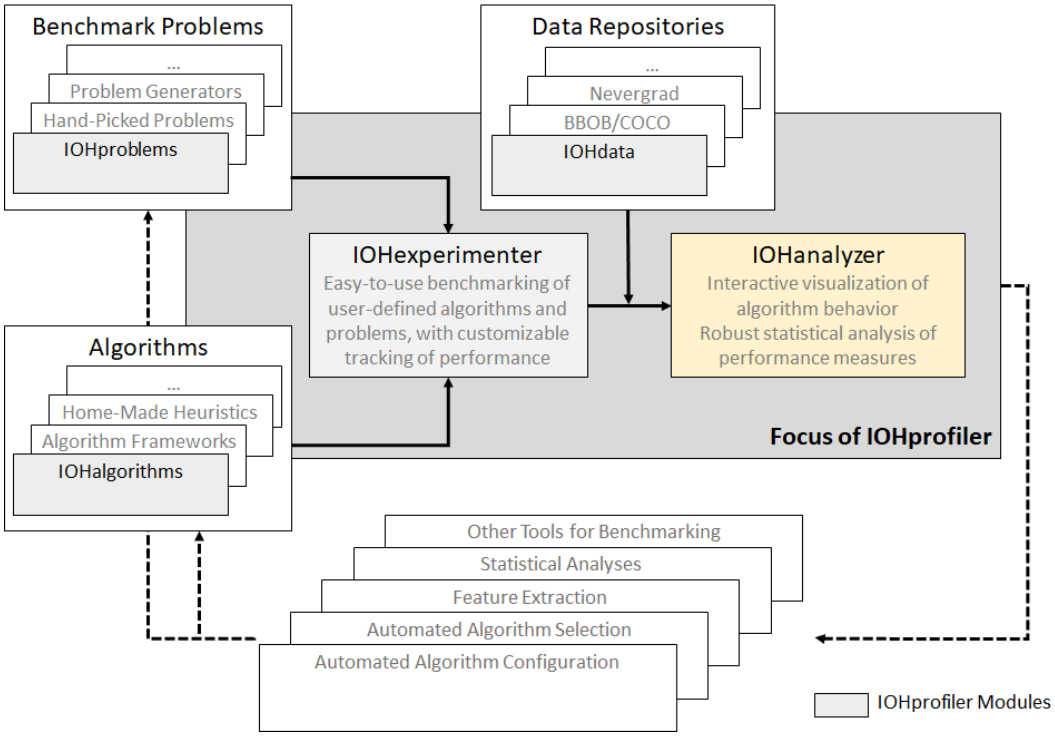


Figure 1: Overview IOH [ioh24].

In Figure 1 a broad overview of IOH is displayed. Using IOHprofiler, this thesis intends to understand how COA works on the BBOB problems in detail on these 24 problems. As seen in the figure, by using a certain algorithm, in this case COA and Random Search, on specific benchmark problems, the BBOB suite, we can use the experimenter to go over each of these problems. Then lastly, we can use the BBOB repository to get data from other algorithms, such as CMA-ES on BBOB, to analyze them with.

4.1.2 BBOB function description

IOH provides 24 BBOB functions with 5 categories among them.

Separable Functions: These functions can be decomposed into smaller sub-problems that can be optimized independently. They are typically easier to optimize because the interaction between variables is minimal. F1 to F5 belong to these functions.

Low or Moderately Conditioned Functions: These functions have some level of interaction between variables but are still relatively straightforward to optimize. They may include functions with moderate conditioning and simple non-separable structures. F6 to F9 belong to these functions.
Ill-Conditioned Functions: These functions present a higher level of difficulty due to their ill-conditioned nature, meaning there is a large disparity in the scales of different directions in the search space. This makes them challenging for many optimizers. F10 to F14 belong to these functions.
Multi-Modal Functions: These functions feature multiple local optima, making it challenging for optimization algorithms to find the global optimum. They test the ability of optimizers to escape local optima and continue searching for better solutions. F15 to F19 belong to these functions.
Weak Structure Functions: These functions have little to no apparent structure, making them particularly difficult to optimize. They test the robustness and generalization ability of optimization algorithms. F20 to F24 belong to these functions.

Abbreviation	Explanation
U	Unimodal: Functions with a single global optimum.
SH	Simple Harmonic: Functions exhibiting harmonic properties.
SE	Separable: Functions where variables can be optimized independently.
NSE	Non-Separable: Functions where variables are interdependent.
M	Multi-modal: Functions with multiple local optima.
R	Rotated: Functions rotated in the solution space.
H	High Conditioning: Functions with a large difference in scale across dimensions.
C	Composite: Functions composed of multiple different function types.

Table 1: Explaining descriptive features of functions.

Each of these categories is designed to test different aspects of an optimizer's performance, from handling simple separable problems to dealing with complex, multi-modal landscapes. By categorizing the benchmark functions in this way, the BBOB test suite can help identify the strengths and weaknesses of COA. It also interesting to learn which functions COA does well at compared to other algorithms. For example, F21 is deemed to be the "best" function to measure the overall performance of an algorithm[HAR+21]. The abbreviations mentioned in Table 1 are typical characteristic properties of functions, which we use to establish what each function does, which we will use to describe the BBOB functions.

4.2 Experimental setup

The experimental setup is designed to ensure that the results obtained are verifiable. For both the analysis on IOH and the additional analysis, we average the results over 5 runs for each function call. For each value of the function call, we take the average result across the 5 runs.

4.2.1 Experiments and analysis using IOHprofiler

The major objective is measuring the performance of COA. To do this, we will be experimenting COA on the 24 different BBOB functions, dimensionalities 2, 3, 5, 10, 20, 30, 40 and 50. The

stopping criteria being 4000 iterations in the main loop, a more detailed view of the main loop can be seen in Section 5.1. Then we will compare the results with CMA-ES, RandomSearch, PSO and ABC. This is done mostly throughout in Section 6. Looking at the ECDF charts we can see the most conclusive evidence of how the algorithm performs overall.

An ECDF (Empirical Cumulative Distribution Function) graph shows the proportion of data points below or equal to a certain value. It plots sorted data values on the x-axis against their cumulative proportion on the y-axis, creating a step-like curve. The Empirical Attainment Function (EAF) estimates the percentage of runs achieving a target value by a certain time. This is what is visible on the y-axis.

F#	**D**	**FS**	**Description**
1	2, 3, 5, 10, 20, 30, 40, 50	1	U, SE
2	2, 3, 5, 10, 20, 30, 40, 50	2	U, SE
3	2, 3, 5, 10, 20, 30, 40, 50	3	M, SE
4	2, 3, 5, 10, 20, 30, 40, 50	4	M, SE, R
5	2, 3, 5, 10, 20, 30, 40, 50	5	U, SE
6	2, 3, 5, 10, 20, 30, 40, 50	6	U, NSE
7	2, 3, 5, 10, 20, 30, 40, 50	7	U, NSE
8	2, 3, 5, 10, 20, 30, 40, 50	8	U, NSE
9	2, 3, 5, 10, 20, 30, 40, 50	9	U, NSE, R
10	2, 3, 5, 10, 20, 30, 40, 50	10	U, NSE, H
11	2, 3, 5, 10, 20, 30, 40, 50	11	U, NSE, H
12	2, 3, 5, 10, 20, 30, 40, 50	12	U, NSE, H
13	2, 3, 5, 10, 20, 30, 40, 50	13	U, NSE, H
14	2, 3, 5, 10, 20, 30, 40, 50	14	U, NSE, H
15	2, 3, 5, 10, 20, 30, 40, 50	15	M, NSE
16	2, 3, 5, 10, 20, 30, 40, 50	16	M, NSE
17	2, 3, 5, 10, 20, 30, 40, 50	17	M, NSE
18	2, 3, 5, 10, 20, 30, 40, 50	18	M, NSE, H
19	2, 3, 5, 10, 20, 30, 40, 50	19	M, NSE, C
20	2, 3, 5, 10, 20, 30, 40, 50	20	M, NSE
21	2, 3, 5, 10, 20, 30, 40, 50	21	M, NSE
22	2, 3, 5, 10, 20, 30, 40, 50	22	M, NSE
23	2, 3, 5, 10, 20, 30, 40, 50	23	M, NSE
24	2, 3, 5, 10, 20, 30, 40, 50	24	M, NSE, C

Table 2: Experimental setup with BBOB functions.

Moreover, Table 2 lists the experimental setup. The results obtained from these experiments will be processed using IOHprofiler to extract relevant data for analysis. This includes the 24 BBOB functions mentioned earlier.

4.2.2 Additional analysis using matplotlib

IOHprofiler provides informative analysis regarding the output and performance of the algorithm. However, the provided implementations from opytimizer and the original algorithm are not directly compatible with IOH. Besides, conducting analysis on other detailed features of the algorithm are not possible. This includes topics such as pack densities, coyote migrations and offspring.

Therefore, it is useful to also conduct external analysis using other data analysis tools. In particular, matplotlib is an easy-to-use python library to create graphs which enables us to better understand these other topics which cannot be done using IOH.

The additional analysis includes examining the difference between my implementation of COA, the implementation of opytimizer and the original implementation of COA in Section 5.3. Moreover, inspecting the correctness of my implementation of COA is done through matplotlib in Section 5.2. As well as different pack migration implementations are measured in Section 5.1.1. Most notably, in Section 6.5 matplotlib is used to measure certain features of the coyote packs.

4.3 Original experiments

In the original publication that describes COA, the researchers also conducted their own analysis on COA. The idea is that by using IOH with my implementation of COA, and other metaheuristic algorithms, we can see that the results are comparable with that of the original publication. Namely, the original publication compares COA with Particle Swarm Optimization (PSO), Artifical Bee Colony (ABC), Symbiotic Organisms Search (SOS), Grey Wolf Optimization (GWO), Bat-inspired Algorithm (BA) and Fire-fly Algorithm (FA). IOH has measured ABC and PSO, so I will compare that with my implementation to see if it is similar to the original publication.

The original setup can be found in the appendix, in Figure B. In this overview we can see that the original publication uses 100 dimensions on certain functions, and includes more functions. In the description it becomes prevalent as to what kind of function they are. These are common functions in IEEE-CEC[SHL+05]. The descriptive meaning of these functions are explained in Table 1.

5 The Algorithm Description

The Coyote Optimization Algorithm is what is known as Swarm Based Intelligence algorithm [NMM20] . The algorithm solves meta-heuristics problems[OK97] . Each coyote acts as an agent which represents a candidate solution. Very briefly, in this algorithm there are a constant number of coyotes, each belonging to a pack. A pack is a subset of the population. The coyotes are able to move between packs and create new ones. The pack has an alpha coyote, the agent with best candidate solution, and a social tendency, the "average" solution of the pack. Each iteration the agents verge towards the alpha coyote and the social tendency of their respective pack. The coyotes age, and old bad performing coyotes get replaced with puppies. These puppies get some attributes of their parents and some mutations. In this chapter, I will describe the algorithm in detail, and some decisions I have made that are unclear in the original publication. As well as the analysis

on why I made these decisions. Moreover, the difference between other implementations will be described. Most notably, the correctness of my implementation will be showcased.

5.1 Pseudocode

In my implementation the only input required is an object from IOH, `ioh.ProblemType`. This describes the dimensionality and objective function that is necessary for the algorithm. The output is the best candidate solution found. Let us use the following notation:

- n: The dimensionality of the search space.
- N_p: count of packs with N_c coyotes each.
- N_c: count of coyotes in the specified pack.
- $\mathbf{x} = (x_1, x_2, \ldots, x_n)$: A solution candidate from $\mathbb{R}^n$.
- $soc_c^{p,t} = \mathbf{x}$: a coyote's c social condition (solution candidate) in pack p, in the tth instant of time.
- $f(\mathbf{x}_i)$: Objective function value of $\mathbf{x}_i$ ($f : \mathbb{R}^n \to \mathbb{R}$).
- lb_{n_i} and ub_{n_i} represent the lower and upper bound on the n_i^{th} dimension.
- $\boldsymbol{\mathcal{U}}(lb_{n_i}, ub_{n_i}) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$: Vector sampled uniformly at random.
- $N_c \cdot N_p =$ population

In algorithm 1 the entire pseudocode of the algorithm is displayed. There are a number of edge cases. For example, albeit unlikely, it is possible that a pack contains zero members. So in all necessary places, we have to be aware of packs of size 0, and not divide by 0. The likelihood of a coyote leaving a pack of size 1 is 0.5% ($P_e = 0.005 \cdot N_c^2$ with $N_c = 1$), and the likelihood of a coyote joining this pack becomes increasingly more likely whenever there are less coyotes in it. Visit the GitHub page to see the full python implementation[vdS].

The amount of times the objective function gets called per iteration is also interesting to learn how many iterations it takes to get a certain target-value. In the initialization the objective function gets called 200 times (as the population size is 200). For each pack, the alpha coyote needs to get determined, in the code we do not call the function each time, but we save the current objective value, so there are no function calls here. Then, for each coyote in this pack, we perform a change and call the objective function. Each iteration a puppy is created and this will also get a function call. The amount of iterations is the following:

$$\text{Function calls} = p + i \cdot apc + i \cdot p,$$

with i being the amount of iterations, p being the population size, and apc being the average pack count over the course of the run.

Algorithm 1 Pseudocode of COA.

Initialization (Alg 2)
$t \leftarrow 1$ ▷ Initial iteration
while $t <$ stopping_condition **do**
  for $p = 1 \rightarrow N_p$ **do** ▷ Iterate over all packs
    $alpha^{p,t} \leftarrow \{soc_c^{p,t} | arg_c(1, 2, \ldots, N_c) min f(soc_c^{p,t})\}$ ▷ Alpha coyote, best of the pack
    $cult^{p,t} \leftarrow median(\{soc_c^{p,t} | arg_c(1, 2, \ldots, N_c)\})$ ▷ Median solution of the pack
    for each coyote c in pack p **do** ▷ Iterate over each coyote in the pack
      pick two random coyotes in p cr_1 and cr_2.
      $\delta_1 \leftarrow alpha^{p,t} - soc_{cr_1}^{p,t}$
      $\delta_2 \leftarrow cult^{p,t} - soc_{cr_2}^{p,t}$
      $new\_soc_c^{p,t} \leftarrow soc^{p,t} + r_1 \cdot \delta_1 + r_2 \cdot \delta_2$ ▷ Update potential solution
      $new\_fit_c^{p,t} \leftarrow f(new\_soc_c^{p,t})$ ▷ Evaluate potential solution
      $soc_c^{p,t+1} \leftarrow \begin{cases} new\_soc_c^{p,t}, & new\_fit_c^{p,t} < fit_c^{p,t} \\ soc_c^{p,t}, & \text{otherwise} \end{cases}$ ▷ Adaptation
    end for
    Birth and death. (Alg 3)
  end for
  Transition. (Alg 4)
  Increment each coyote's age by 1.
end while

5.1.1 Ambiguous coyote migrations

The original publication is unclear about how coyotes choose which pack to join. It explains the probability of a coyote leaving a pack but not the process of how they join a new one. I tested two different implementations: one where the likelihood of joining any pack or starting a new one is equal, and another where the likelihood of joining a pack is inversely related to the number of coyotes already in that pack, causing the pack sizes to be roughly even.

In Figure 2 I compare the performance of COA with the two different ways coyotes migrate. The results show how far off the found result is off the optimum in dimension 20 with 4000 iterations. My implementation of COA has the pack sizes even. The orange line indicates where the chances are equal for all packs. From these results we can indicate that both solutions are worthwhile to try, with a slight preference towards an even size among the packs.

Based on these findings, I have decided to continue with the evenly distributed pack sizes for further experimentation.

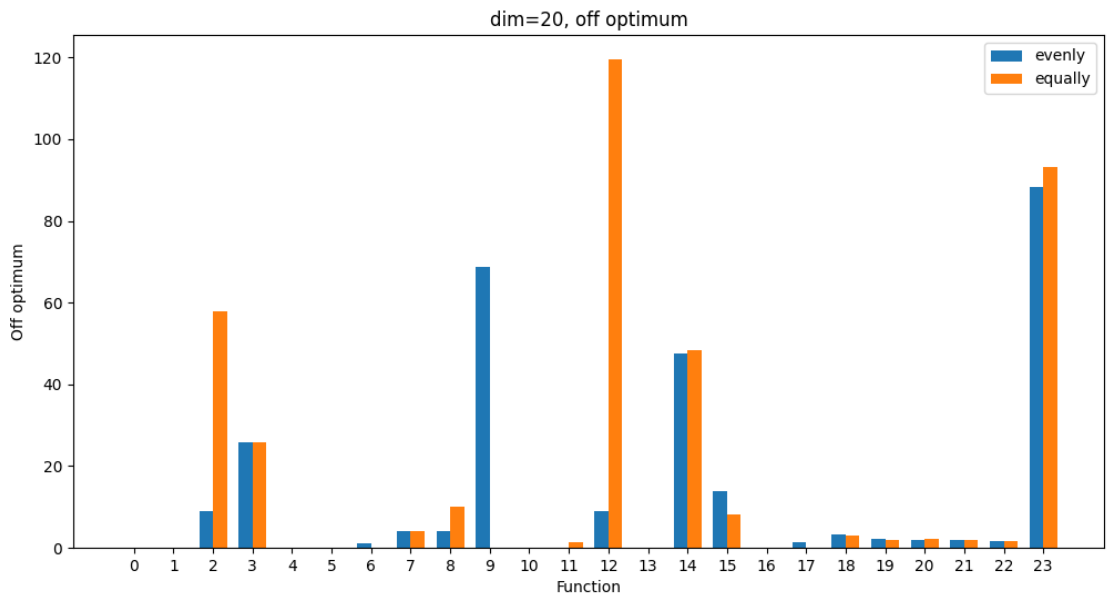


Figure 2: Evenly versus equally

5.2 Correctness of COA

Of course, correctness of the implementation of COA is crucial for further research, and understanding of the algorithm. To determine this, we will look at a straightforward sphere function in 2 dimensions and compare the results to an already established meta-heuristic algorithm, CMA-ES. This algorithm is used as a baseline to measure meta heuristic algorithms[Han06].

5.2.1 COA on the 2d Sphere function

As seen in Figure 3 it is obvious that it is converging towards the optimum. Each dot represents a coyote's fitness. In merely 10 iterations almost all coyotes have converged and at least one has found the optimum.

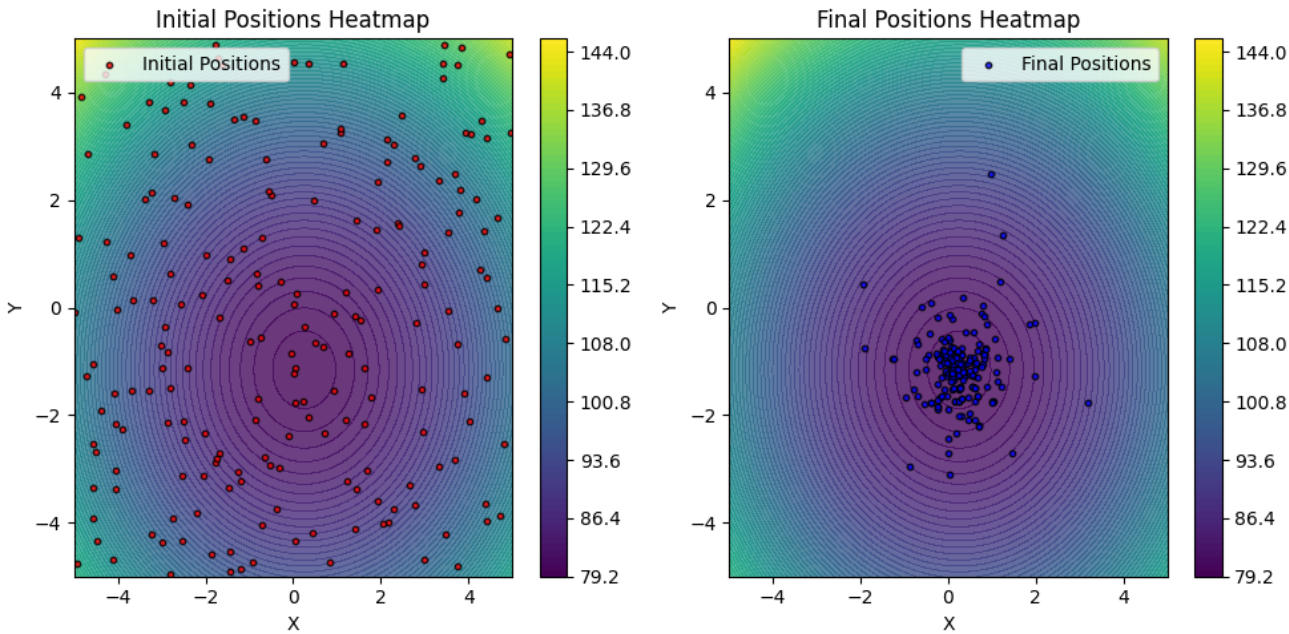


Figure 3: COA on 2D sphere function in 10 iterations.

5.2.2 a-CMA-ES and COA

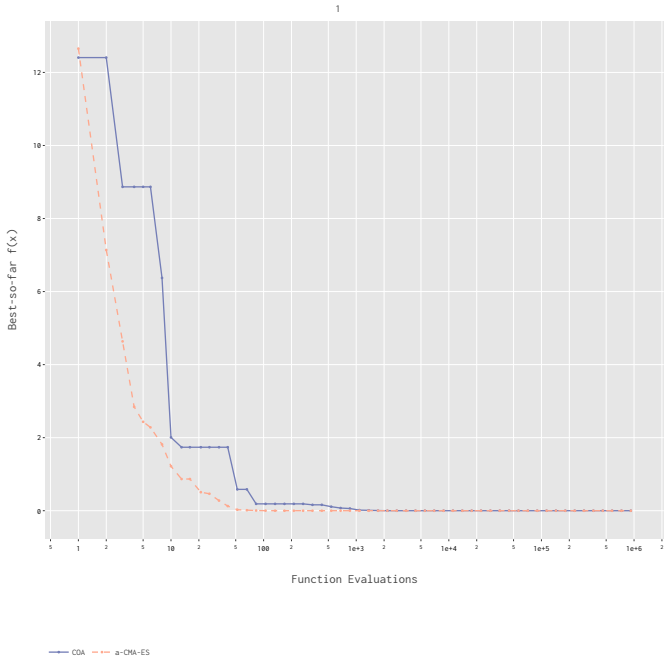


Figure 4: a-CMA-ES vs COA on 2D sphere function.

In Figure 4 COA competes against a-CMA-ES on the 2d sphere function, and it is also able to obtain the optimum. This showcases that the implementation of COA is correct in the sense

that is able to converge and reach the optimal solution, similar to an established algorithm like a-CMA-ES. This validation confirms the correctness of the COA implementation by demonstrating its effectiveness on a well-known benchmark problem.

5.3 Different Implementations

As mentioned in Chapter 5.1.1, there are some unclear aspects in COA, resulting in a unique coyote migration method. Additionally, the original publication is ambiguous about whether certain variables are generated in each iteration or as a single random value. In Algorithm 2, the variables r_1 and r_2 are used to adapt the candidate solutions. In my implementation, these variables are generated randomly once, whereas in Opytimizer and the original implementation, they are generated anew for each adapting coyote. Furthermore, the original publication used a different stopping condition which made the dependency based on the dimension size. The more dimensions, the more iterations. In reality, this is not very important because different research can be based on a different budget.

Table 3: Parameter table of my COA and original COA.

Parameter	Description	Default Value (Original)	My default Value	Accepted Range
Population Size	Total coyotes	200	200	1 - Unlimited
Max pack size	$\sqrt{\texttt{population}}$	14	14	1 - population
Stop condition	Dimension dependant	D·10,000	4000	1 - Unlimited
P_s	Scatter probability	$\frac{1}{N}$	$\frac{1}{N}$	0 - 1
P_a	Association probability	$\frac{1-P_s}{2}$	$\frac{1-P_s}{2}$	0 - 1
P_e	Eviction probability	$0.005 \cdot N_c^2$	$0.005 \cdot N_c^2$	0 - 1
Initial pack size	Randomly chosen	$U(5, 10)$	$U(5, 10)$	1 - population

In Table 3 a short overview is given of some optional parameters. The only difference is the stopping criteria, which is different, because we wanted to remain at slightly under 1 million function calls. The amount of function calls is determined in Section 5.1.
With all these different implementations, one has to know how they perform to one another. I have tested the different versions using the sphere function once again.

In Figure 5 we see 4 different graphs with each the 3 algorithms in question. The variable that is being changed is the dimension. Each run has 100 iterations in the main loop. It becomes apparent that the opytimizer implementation outperforms COA and the original COA. My implementation is in between the two. However, they all perform very similar in the grand scheme.

6 Experiments and Findings

There are multiple interesting topics to discuss. Namely, the performance from COA on the BBOB problems. It is interesting to see at which functions BBOB can excel at. Secondly, the original publication conducted it's own analysis on the performance of COA, and it is very interesting to compare it to my implementation, so we will compare these two. Moreover, it is interesting to

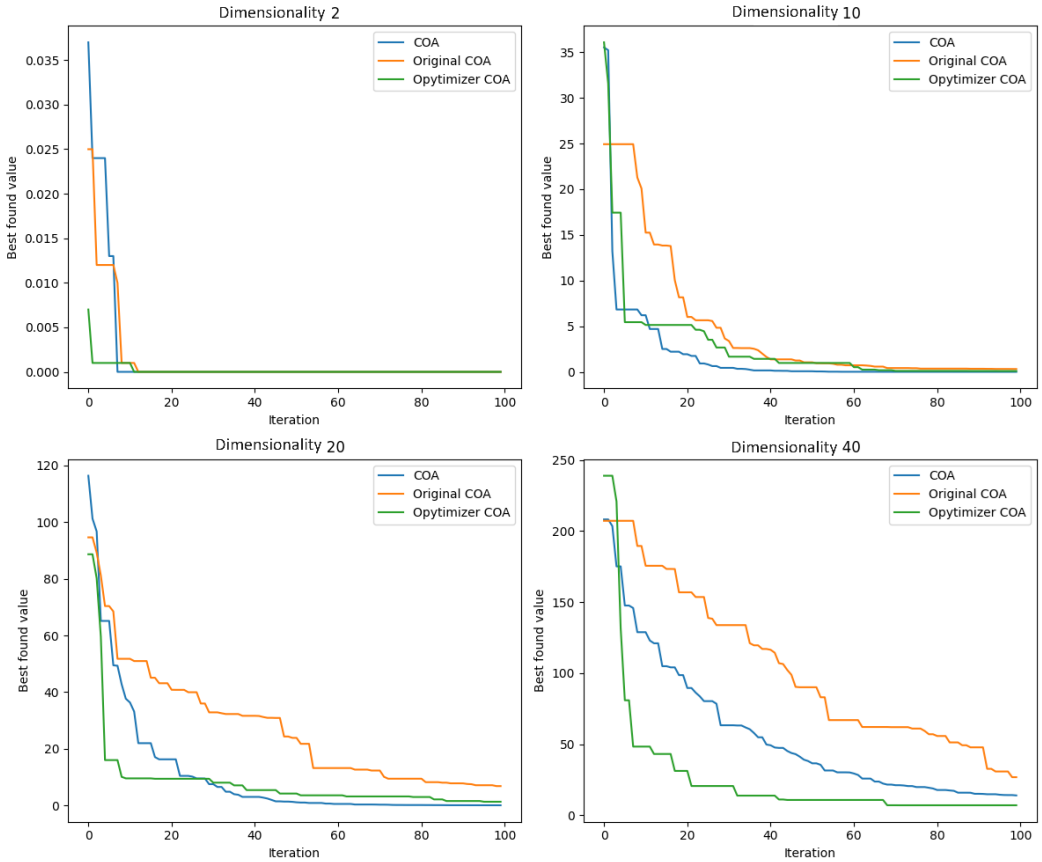


Figure 5: Different dimensionality on the sphere function with original COA, opytimizer COA and my COA.

look at the dimensionality of COA, and and metaheuristic algorithm, as the problems become increasingly more difficult. Besides, looking at established metaheuristic functions, CMA-ES and Random Search, give us a good view as to how well COA performs in general. Lastly, investigating how the algorithm explores and exploits is vital to understand COA to predict how well it does in other metaheuristic problems. The parameter settings selected in the experiment are seen in Table 3, with the experimental setup from Section 4.2.

6.1 Swarm Based Intelligence on BBOB

As mentioned before, COA is a swarm based intelligence much like PSO. These types of algorithms work very similar on certain BBOB functions in particular.

To show the resembling behaviour of COA and PSO, take a look at Figure 6. In this image we can see the overall best found algorithm, BIBOP-CMA-ES, RandomSearch, PSO and COA compete against each other in dimension 40 over all the BBOB functions. It becomes apparent that PSO and COA have a very similar learning curve. However, COA seems to perform significantly better on F21. This displays the superiority of COA compared to PSO. When looking at the large-scale benchmarking performance, we can conclude that these findings are very similar [VDW+24].

6.2 Difficulty in Dimensions

In metaheuristics it is important to determine the value of the algorithm in each dimension. In general, the higher the dimension, the more difficult it becomes. However, does the improvement rate scale similarly to other algorithms, or does it get comparatively worse or better than other algorithms?

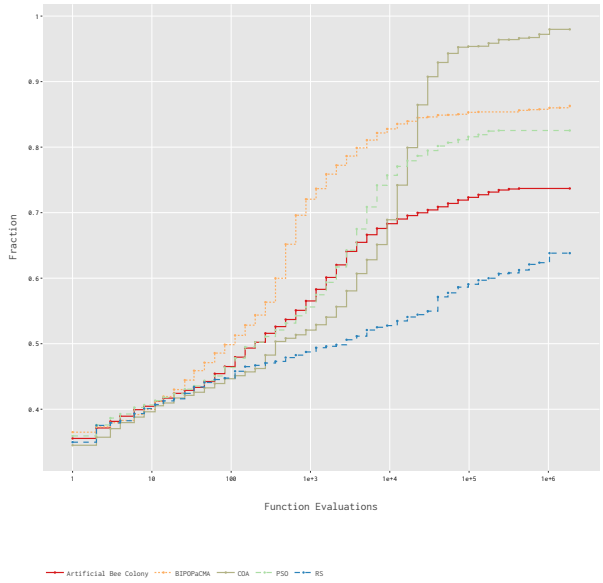
In general, swarm based intelligence scores well in low dimensionality[HSCS19]. Take a look at Figure 6. In this graph we can see the ECDF output of CMA-ES, COA, ABC, PSO and RS across 4 different dimensionsalities.

In Figure 6 we can see that BIPOP-CMA-ES gets beaten by COA in dimensionalities 2 and 5. However, by increasing the dimensionality COA quickly becomes worse than the benchmark standard. Moreover, as the dimensionality increases, COA begins to exhibit behavior more characteristic of its family of meta-heuristic algorithms, such as PSO and ABC

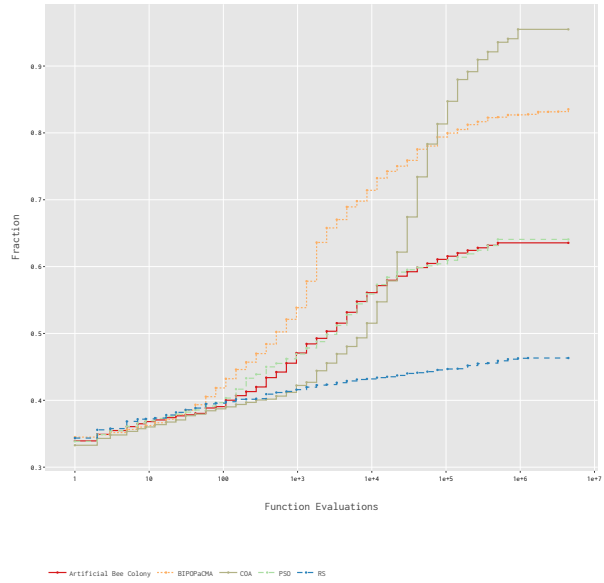
6.3 Comparison with Original COA

As mentioned in Section 4.3, the original publication of COA has benchmarks against ABC and PSO which is also possible in IOH. Moreover, the original benchmark on COA uses F71, which is described as the a non-separable and multimodal benchmark named Griewank's plus the Rosenbrock's Function. F71 is also accessible in BBOB, but it is labeled as F19c. However, the original publication runs this experiment in 100 dimensions, but IOH has it available up to 20.
This issue is not a problem in general, because fortunately, ABC, COA and PSO are all swarm based intelligence. As mentioned in Section 6.2, we can see that swarm based intelligence intend to do well in lower dimensionalities and increasingly get worse in higher dimensionalities, compared to CMA. So, the improvement rate of all these algorithms is roughly the same in these high dimensionalities. Comparing the improvement curve of the two different outputs, we can see that this still holds true. Both in IOH, and in the original publication, COA and PSO perform better than ABC, with a slightly faster improvement rate. The overall learning trend is still the same.

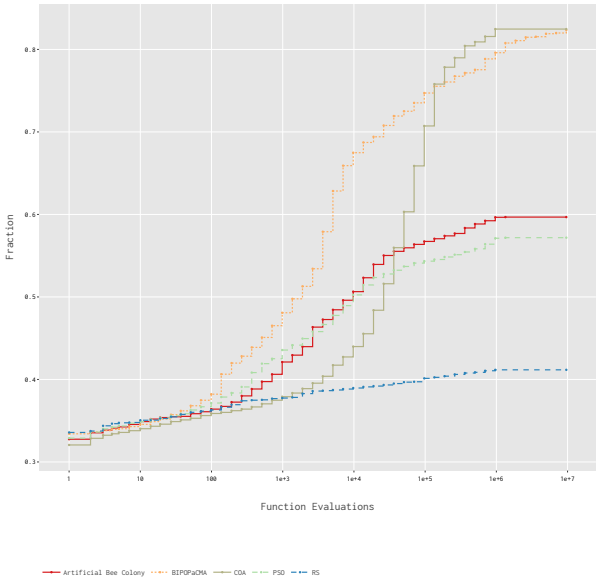
Besides, when we look at the Large-Scale benchmarking research from LIACS, COA outperforms ABC and PSO in average loss over all 24 BBOB functions for each algorithm. This is also confirmed in Figure 6 and Figure 7 as COA performs better than the other swarm based intelligence algorithms on the most difficult BBOB function, F21.



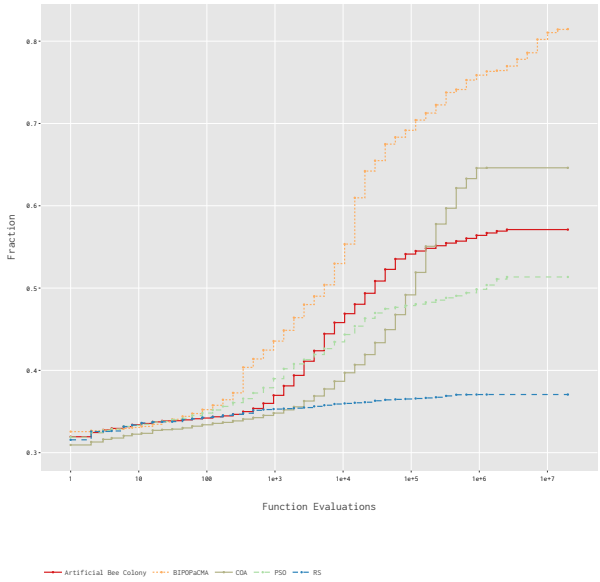
(a) Dimensionality 2



(b) Dimensionality 5

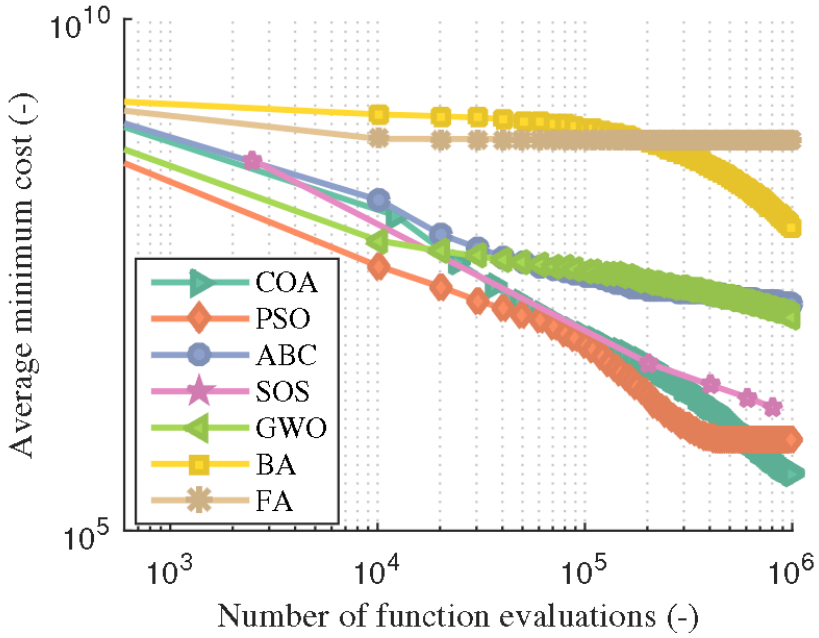


(c) Dimensionality 10



(d) Dimensionality 20

Figure 6: ECDF performance on CMA-ES(orange), COA(brown), ABC(red), PSO(green) and RS(blue) with dim 2,5,10,20.



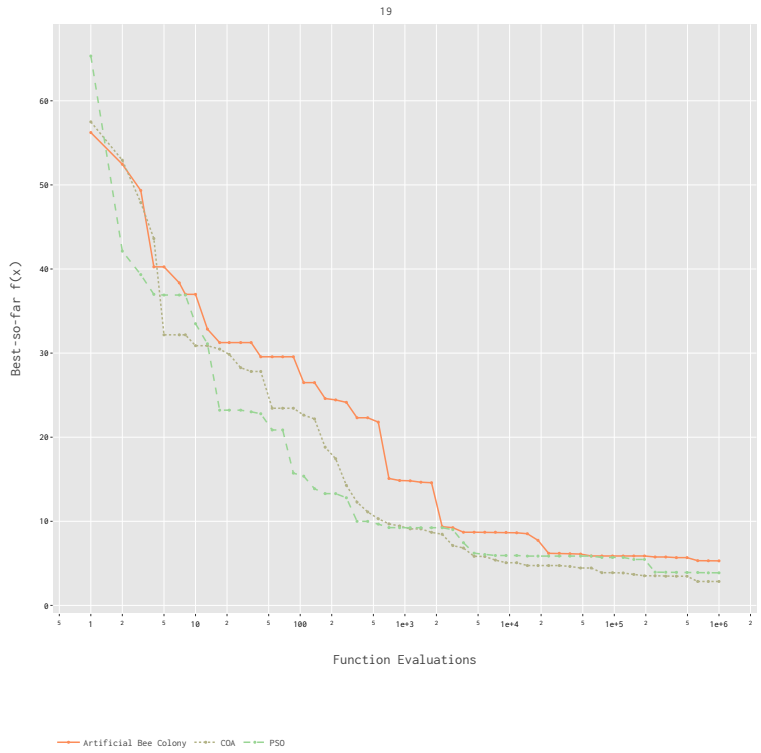


Figure 7: F19 with COA, PSO, ABC from the original paper(top) and on IOH(bottom).

6.4 Benchmarking COA

To benchmark ones meta-heuristic algorithm, we need to measure it's performance on functions, against other meta-heuristic algorithms. It is of course mandatory to compare any algorithm strictly with BIPOP-CMA-ES and Random Search. These two algorithms are the standard benchmarking tool to compare any meta-heuristic algorithm with, as CMA-ES is considered to be the most well-rounded meta-heuristic algorithm, and Random Search, because it is the most simplistic.

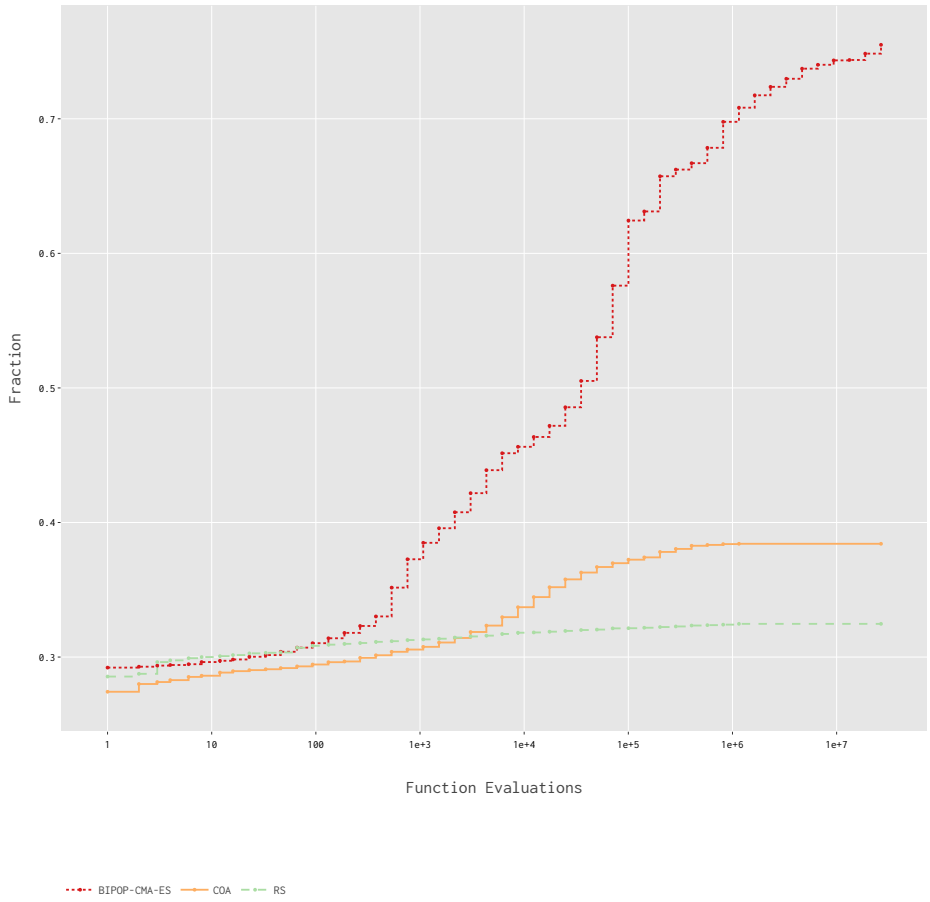


Figure 8: Dimensionality 40 with RS(green), COA(orange), and BIPOP-CMA-ES(red).

If we look at Figure 8 it is apparent that COA is not great. In the first 1000 iterations it even performs worse than Random Search, but starts to pick up from there. Note that this is in dimensionality 40, whereas the other ECDF graphs were in 20. This is because IOH does not support dimensionality 40 for ABC nor PSO.

It is remarkable that we can see in Figure 9, despite the notably poor performance in Figure 8, COA excels in certain BBOB functions. In high dimensionalities the overall performance gets worse, but in some functions COA can still compete with BIBOP-CMA-ES. The Sphere Function F1 is of course very easy, so this performance is not surprising. However, if we look at F21 and F22 in particular, very similar functions, COA is still able to do exceptionally well. And then in F23, COA

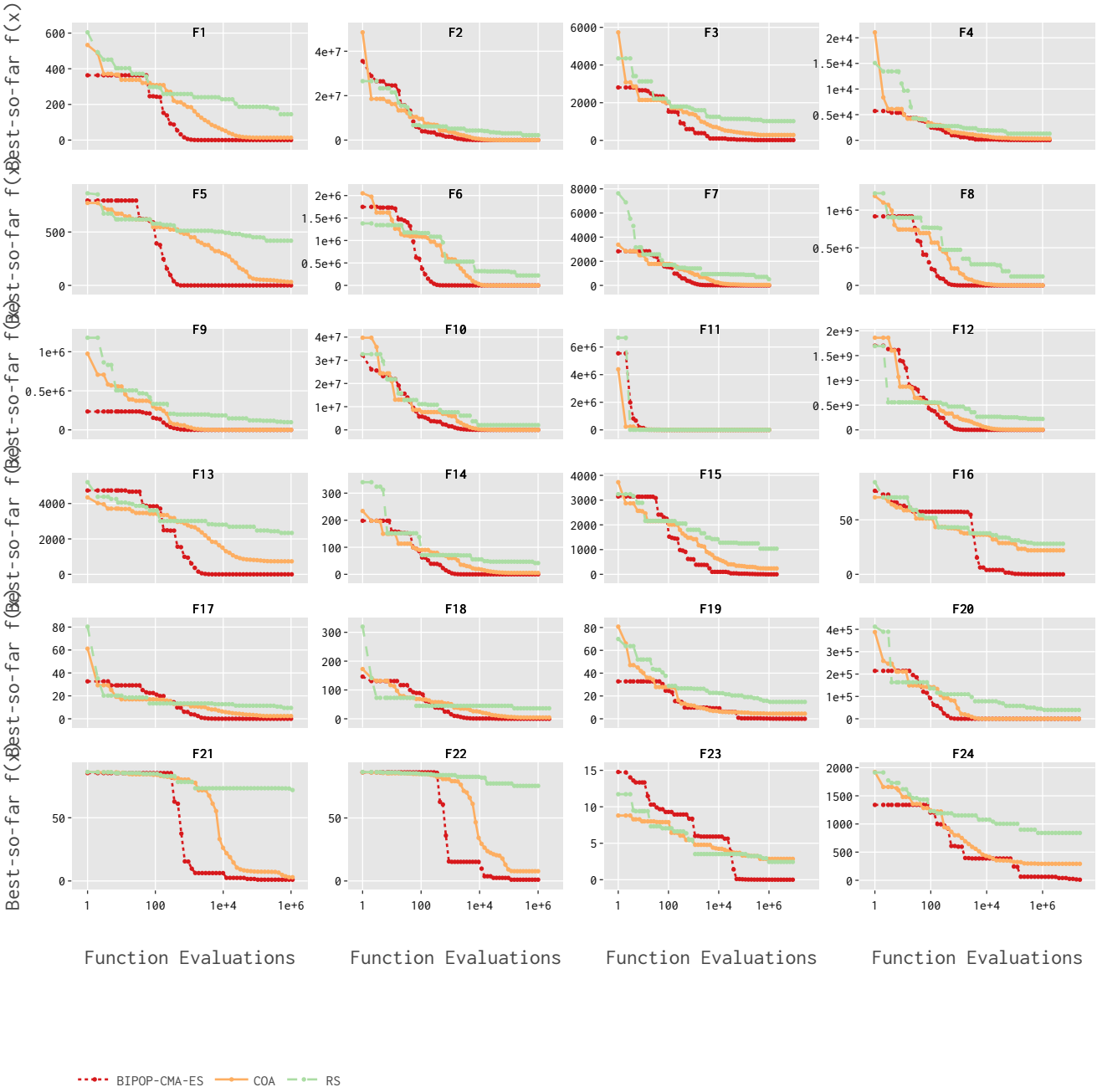


Figure 9: Dimensionality 40 with all BBOB functions on CMA-ES(red), COA(orange), RS(green).

performs worse than Random Search. This shows the difficulty of the BBOB functions in higher dimensionalities and that certain functions have a speciality of solving certain problems, despite CMA-ES' overall dominance.

6.5 Explore and Exploit

In any meta-heuristic algorithm we are curious how it explores and exploits. In general, it is good to first explore and then exploit, when a reasonably good, ideally the best, optimum has been found. The way COA explores and exploits is with packs. In the start all coyotes are scattered and they verge to the cultural tendency and alpha coyote. In the midst of this process they might find better optima and become the alpha themselves. When the packs are "stacked" in the search space, the exploitation takes over and the entire pack is looking for the local optimum. We can see how a pack converges to certain local optima by looking at pack densities, and we can also take a look if the packs converge to different local optima in the search space.

6.5.1 Pack densities and pack convergence

Measuring the pack densities is key to understand how the algorithm behaves. There are two specific qualities to look at: the pack densities within the pack itself, meaning the average distance of all coyotes from one another, and the pack's average location in the solution space to the best found location. I will be looking at F21 and F23 in dimensionality 40 in particular. These observations are intriguing because, as noted in Section 6.4, F21 is particularly challenging, yet COA performs exceptionally well on it. Conversely, F23 is also difficult, but COA performs worse than Random Search on this function.

Each line in Figure 10 to 13 represents a pack of coyotes in the search space.

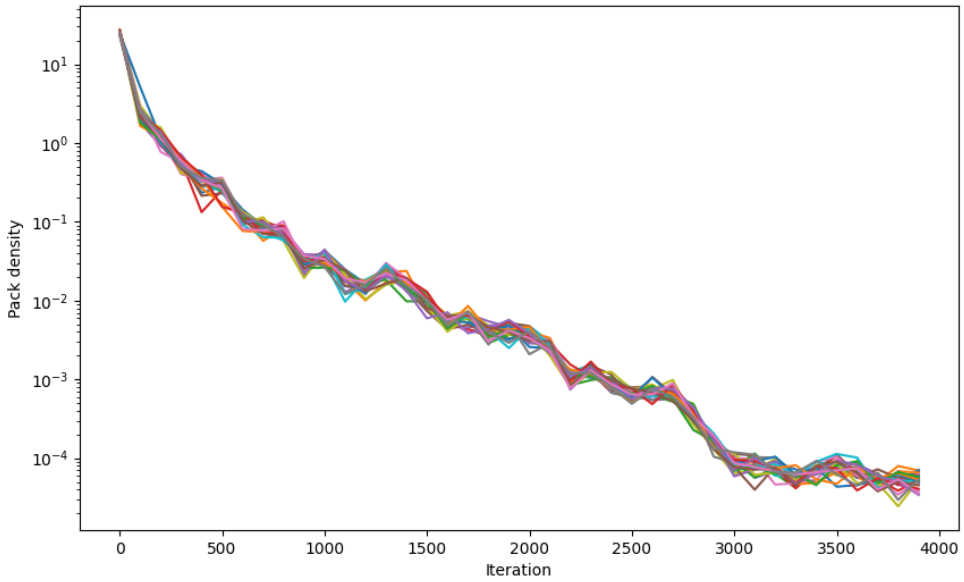


Figure 10: Pack densities of all packs on F21.

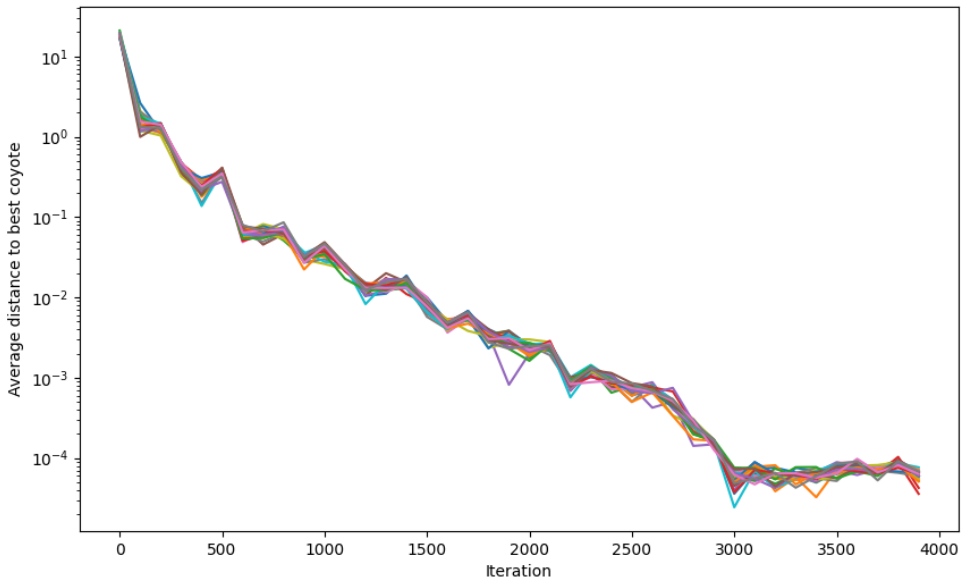


Figure 11: Average pack location's distance to best coyote for F21.

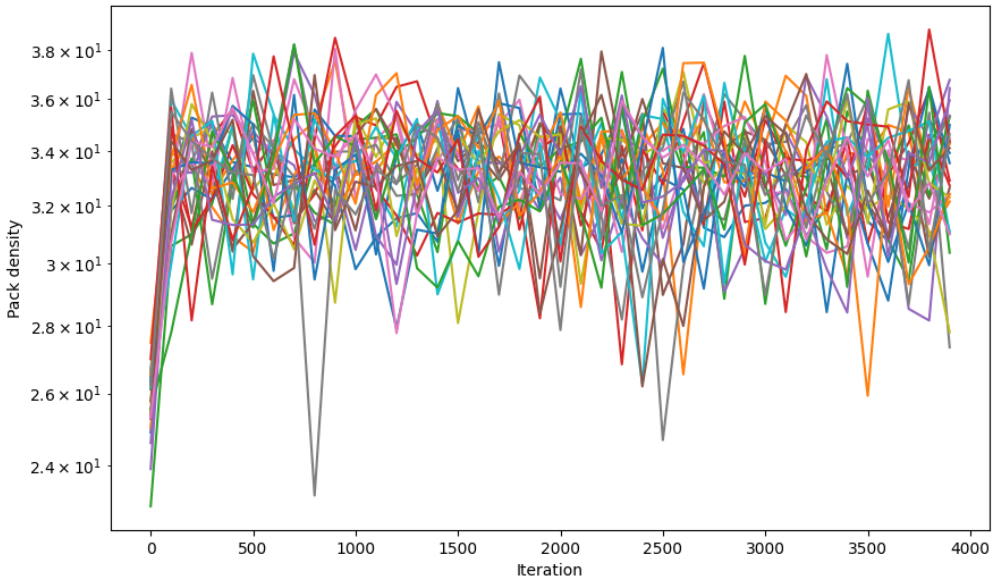


Figure 12: Pack densities of all packs on F23.

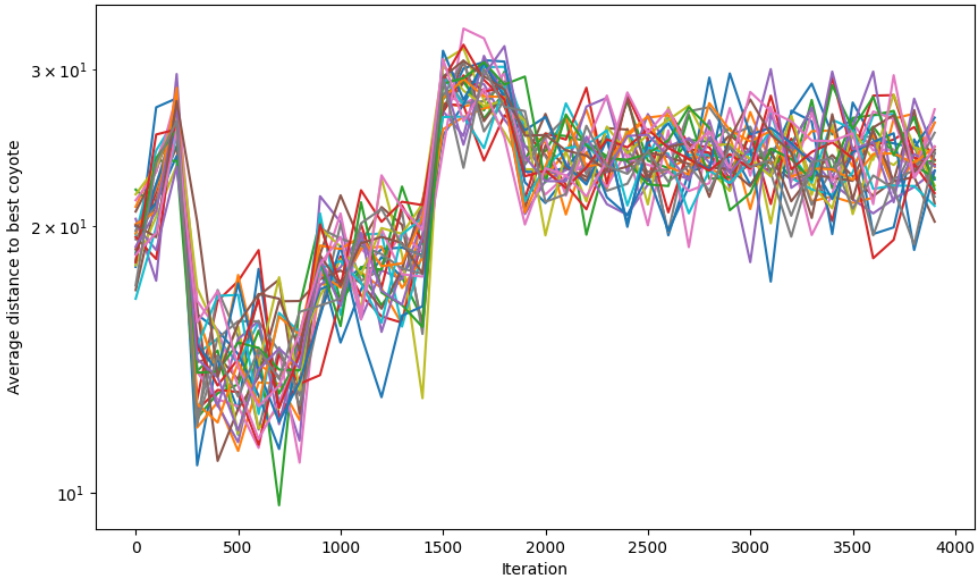


Figure 13: Average pack location's distance to best coyote for F23.

When observing figures 11 and 10 we notice a strange trend: the pack density, meaning the average distance from all coyotes with each other, tends to be the same as the pack's average location compared to the best coyote's location (over all packs). This means that each pack, independently, acts the same as they do not know where the best coyote is, except for the pack with this coyote. In reality, it means that all packs are getting stuck in the same local optimum, and are converging roughly the same. Using multiple different seeds yields the same results. This is quite strange, as the idea with Gallagher's Gaussian function is that peaks are quite randomly placed in the search space

However, if we look at figures 13 and 12, the results seem to be stochastic. The average pack density is very inconsistent, but the distance of the average location for each pack in the search space to the optimal solution seems to be changing. This is because, over time and through chance, a new location for the best coyote is found and the candidate solutions are still scattered around in the search space with no real exploitation. Looking at the algorithm in Section 5 and F23, the Katsuura function[HAR+21], we can understand why things go wrong. When the coyotes verge to their alpha coyote and social tendency, they each meet new local optima and so the packs stay scattered. It comes down to random search at that point, as there are no real patterns in the search space.

This means that COA starts exploitation once the coyote pack densities narrow down, and if they stay scattered, the algorithm will remain stochastic and in its exploratory state.

6.5.2 Converging to local optima

In Section 6.5.1 we can see that in F21 the coyotes are converging towards an optimum, and that in F23 the algorithm seems stochastic.

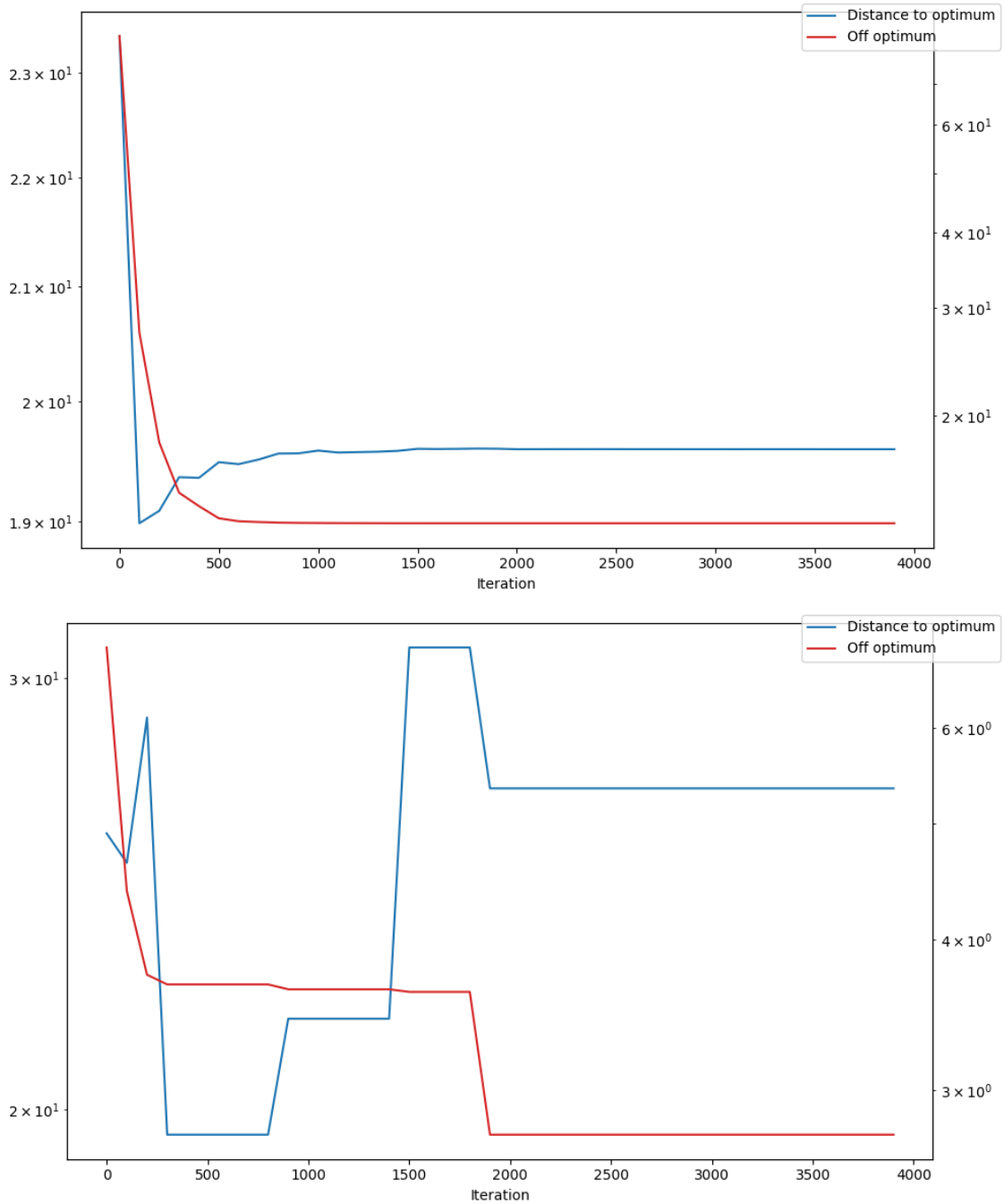


Figure 14: F21 vs F23

In Figure 14 we see the same function output, but here we look at the best coyote's distance to the optimal location on the blue line, and the best found solution on the red line, with F21 on the left hand side, and F23 on the right hand side. Notice that in F21, best found solution is moving away from the optimum. Clearly, this means the algorithm is stuck in a local optimum and will never find the global optimum, because the distance of the optimum is still reasonably far away. For F23, the graph resembles Figure 13. This is because the distance of all coyotes is scattered through the search space and the distance to the best coyote and the best found optimum is equally randomly placed in the search space.

7 Conclusions and Further Research

The implementation and evaluation of the Coyote Optimization Algorithm presented several key insights. Implementing COA was a fairly tricky, because the original publication did not serve much pseudcode, but mainly mathematical formulas which had some small ambiguity. The implementation ended up being slightly different than the implementation from the original authors and of another implementation from opytimizer, but still performed equally well, if not better. The biggest difference being the coyote migration method, while the original publication does not clarify how it was done, I implemented an evenly distributed way of migration.

When experimenting with COA there were some interesting findings. Most notably, COA demonstrated very strong performance in low-dimensional spaces, even outperforming the traditional optimization algorithms overall. Especially on the BBOB functions F21 and F22, which seem sporadic, but COA can consistently in even high dimensionalities outperform CMA-ES. Another observation is that COA starts behaving like any other swarm based intelligence algorithm in higher dimensions, suggesting that regardless of the way of how agents change, they act similar across the different algorithms.

In the EC-Bestiary collection it is relevant to know which algorithms are worthwhile in the world of meta-heuristic problem solving, and COA definitely has a place to be relevant, because of it's dominance in low-dimensionality. It is very similar to some other swarm based intelligence algorithms such as PSO and ABC, but it still has it's unique characteristics in the form of subsets of agents representing packs.

If this endeavour on COA was picked up again, it would definitely be interesting to adjust the algorithm such that each pack has a unique characteristic in the way they explore and exploit, because it appears that each pack independently from one another act very similar, causing each pack to find the optimum, or for none of them to get close to an optimum.

References

[AAPV19] Stavros P Adam, Stamatios-Aggelos N Alexandropoulos, Panos M Pardalos, and Michael N Vrahatis. No free lunch theorem: A review. *Approximation and optimization: Algorithms, complexity and applications*, pages 57–82, 2019.

[BBDB+20] Thomas Bartz-Beielstein, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, William La Cava, Manuel Lopez-Ibanez, et al. Benchmarking in optimization: Best practice and open issues. *arXiv preprint arXiv:2007.03488*, 2020.

[CA23] Felipe Campelo and Claus Aranha. Lessons from the evolutionary computation bestiary. *Artificial Life*, 29(4):421–432, 2023.

[Cam24] Felipe Campelo. EC-Bestiary: An Evolutionary Computation Bestiary. https://github.com/fcampelo/EC-Bestiary, 2024. [Online; accessed 23-February-2024].

[CK17] Amrita Chakraborty and Arpan Kumar Kar. Swarm intelligence: A review of algorithms. *Nature-inspired computing and optimization: Theory and applications*, pages 475–494, 2017.

[CP14] Min-Yuan Cheng and Doddy Prayogo. Symbiotic organisms search: a new metaheuristic optimization algorithm. *Computers & Structures*, 139:98–112, 2014.

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

[GHdRP19] Douglas Rodrigues Gustavo H. de Rosa and João P. Papa. Opytimizer: A nature-inspired python optimizer, 2019.

[Han06] Nikolaus Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pages 75–102, 2006.

[HAR+21] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36:114–144, 2021.

[HI24] Haoran-Ian. Random search. https://github.com/haoran-ian/HandsonBachelorThesis/blob/master/random_search.py, 2024. [Online; accessed 14-June-February-2024].

[HSCS19] Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. On the exploration and exploitation in popular swarm-based metaheuristic algorithms. *Neural Computing and Applications*, 31(11):7665–7683, 2019.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[ioh24] Ioh analyzer. https://iohprofiler.github.io/, 2024. [Online; accessed 14-June-February-2024].

[Jkp] Jkpir. Jkpir/coa: A new metaheuristic for global optimization problems proposed in the ieee congress on evolutionary computation (cec), 2018.

[Kar10] Derviş Karaboğa. 'artificial bee colony algorithm. *scholarpedia*, 5(3):6915, 2010.

[MML14] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.

[NHB21] Raymond Ros Olaf Mersmann Tea Tušar Nikolaus Hansen, Anne Auger and Dimo Brockhoff. Coco: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[NMM20] Nadia Nedjah, Luiza De Macedo Mourelle, and Reinaldo Gomes Morais. Inspiration-wise swarm intelligence meta-heuristics for continuous optimisation: a survey-part ii. *International Journal of Bio-Inspired Computation*, 16(4):195–212, 2020.

[OK97] Ibrahim H Osman and James P Kelly. Meta-heuristics theory and applications. *Journal of the Operational Research Society*, 48(6):657–657, 1997.

[PC18] Juliano Pierezan and Leandro Dos Santos Coelho. Coyote optimization algorithm: a new metaheuristic for global optimization problems. In *2018 IEEE congress on evolutionary computation (CEC)*, pages 1–8. IEEE, 2018.

[SHL+05] Ponnuthurai N Suganthan, Nikolaus Hansen, Jing J Liang, Kalyanmoy Deb, Ying-Ping Chen, Anne Auger, and Santosh Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. *KanGAL report*, 2005005(2005):2005, 2005.

[vdS] Ivar van der Spoel. GitHub - ivarvdspoel/bsc_coa: Baschelor Thesis on the Coyote Optimization Algorithm — github.com. https://github.com/ivarvdspoel/bsc_coa. [Accessed 23-06-2024].

[VDW+24] Diederick Vermetten, Carola Doerr, Hao Wang, Anna V Kononova, and Thomas Bäck. Large-scale benchmarking of metaphor-based optimization heuristics. *arXiv preprint arXiv:2402.09800*, 2024.

[WTL18] Dongshu Wang, Dapei Tan, and Lei Liu. Particle swarm optimization algorithm: an overview. *Soft computing*, 22:387–408, 2018.

[Yan09] Xin-She Yang. Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169–178. Springer, 2009.

[Yan10] Xin-She Yang. A new metaheuristic bat-inspired algorithm. In *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pages 65–74. Springer, 2010.

Appendices

A COA in Detail

Algorithm 2 initialization

```
population ← 200                                   ▷ Hard value for population size
while i = 1 < population do
    N_c ← randint(5, 10)                           ▷ Hard value initial N_c ∈ [5, 10]
    if N_c + i > population then                   ▷ Pop does not exceed 200
        N_c := 200 − i
    end if
    i ← i + N_c
    p ←create_new_pack()                           ▷ New empty pack
    for j = 1 → N_c do
        soc_j^{p,0} := U(lb_{n_i}, ub_{n_i})       ▷ Random initial values
        p.add(soc_c^{p,0})                         ▷ Add coyote to pack
    end for
    world.add(p)                                   ▷ Add pack to world
end while
for all coyotes c in pack p do
    f(soc_c^{p,0})                                 ▷ Verify Coyote's adaptation
end for
r_1, r_2 ∈ [0, 1]                                  ▷ uniformely randomly generated.
P_s ← 1/n                                          ▷ n is the dimensionality
P_a ← (1−P_s)/2
stopping_condition ← 4000                          ▷ Any number of iterations as prefered.
```

Algorithm 3 birth and death

For each pack p ▷ Call this function in pack loop in Alg 1
$\varphi \leftarrow |p|$ ▷ Count of coyotes in pack
$j_1 \leftarrow randint(0, N-1)$
$j_2 \leftarrow randint(0, N-1)$ ▷ Random dimension number
$r_1 \leftarrow randint(0, \varphi)$ ▷ Random coyote in pack p
$r_2 \leftarrow randint(0, \varphi)$
$pup_j^{p,t} \leftarrow$ new pup
for $i = 1 \rightarrow N$ **do** ▷ Iterate over dimensions
    $rnd_i \in [0, 1]$ ▷ Uniformely random generated
    if $rnd_i < P_s$ or $i = j_1$ **then**
        $pup_i^{p,t} \leftarrow soc_{r_1,i}^{p,t}$ ▷ Pup dim i value
    else if $rnd_j \geq P_s + P_a$ or $i = j_2$ **then**
        $pup_i^{p,t} \leftarrow soc_{r_2,i}^{p,t}$ ▷ Pup dim i value
    else
        $pup_i^{p,t} \leftarrow R_i \in [lb_i, lb_i]$ ▷ Uniformly random generated number between bounds
    end if
end for
$\omega \leftarrow \{soc_c^{p,t} | arg_c(1, 2, ..., N_c) f(soc_c^{p,t}) > f(pup_j^{p,t})\}$ ▷ Coyotes with worse solution than new pup
if $\varphi = 1$ **then** ▷ Singular coyote in pack
    kill the only coyote in pack p, and add pup.
else if $\varphi > 1$ **then** ▷ More than 1 coyote in pack
    Oldest coyote in ω dies, and add pup to pack p.
else ▷ Pack size is 0
    The pup dies.
end if

Algorithm 4 Transition Evenly

```
for For each pack p in world do
    for each coyote c in pack p do
        N_c ← |p|
        P_e ← 0.005 · N_c^2                                   ▷ Compute eviction chance
        if P_e > U(0,1) then                                  ▷ Uniformly random generated
            Remove coyote c from pack p
            inversions ← {1/(N_c+1) | over all N_p}           ▷ High N_c lower number
            total ← ∑ inversions                              ▷ Total of all packs
            probabilities ← {inversion/total | for inversion in inversions}   ▷ Low N_c high number
            new_pack_nr ∼ Multinomial(N_p, p = probabilities) ▷ Low N_c likelier
            if N_c < √population then                         ▷ Max pack size
                Add coyote c to pack with pack number new_pack_number
            else
                Coyote c will form a new pack   ▷ Form new pack p, put c in it, add p to world.
            end if
        end if
    end for
end for
```

B Original Experimental Setup

F#	*D*	FS	Description
1, 2, 3	30, 50, 100	1	U, SH, SE
4, 5, 6	30, 50, 100	2	U, SH, NSE
7, 8	30, 50	3	U, SH, R, NSE
9, 10, 11	30, 50, 100	4	U, SH, NSE
12, 13, 14	30, 50, 100	5	U, NSE
15, 16, 17	30, 50, 100	6	SH, M, NSE
18, 19	30, 50	7	SH, R, M, NSE
20, 21	30, 50	8	SH, R, M, NSE
22, 23, 24	30, 50, 100	9	SH, SE, M
25, 26	30, 50	10	SH, R, M, NSE
27, 28	30, 50	11	SH, R, M, NSE
29, 30, 31	30, 50, 100	12	M, NSE
32, 33, 34	30, 50, 100	13	M, NSE
35, 36	30, 50	14	SH, R, M, NSE
37	30	15	SH, SE, C, H, M
38	30	16	SH, R, C, H, M, NSE
39	30	17	SH, R, C, H, M, NSE
40	30	18	SH, R, C, H, M, NSE
41	30	19	SH, R, C, H, M, NSE
42	30	20	SH, R, C, H, M, NSE
43	30	21	SH, R, C, H, M, NSE
44	30	22	SH, R, C, H, M, NSE
45	30	23	SH, R, C, H, M, NSE
46	30	24	SH, R, C, H, M, NSE
47	30	25	SH, R, C, H, M, NSE
48, 49, 50	30, 50, 100	1	U, R, NSE
51, 52, 53	30, 50, 100	2	U, R, NSE
54, 55, 56	30, 50, 100	3	SH, R, M, NSE
57, 58, 59	30, 50, 100	4	SH, R, M, NSE
60, 61, 62	30, 50, 100	5	SH, R, M, NSE
63, 64, 65	30, 50, 100	6	SH, R, M, NSE
66, 67, 68	30, 50, 100	7	SH, R, M, NSE
69, 70, 71	30, 50, 100	8	SH, R, M, NSE
72, 73, 74	30, 50, 100	9	SH, R, M, NSE
75, 76, 77	30, 50, 100	10	H, M, NSE
78, 79, 80	30, 50, 100	11	H, M, NSE
81, 82, 83	30, 50, 100	12	H, M, NSE
84, 85, 86	30, 50, 100	13	C, M, NSE
87, 88, 89	30, 50, 100	14	C, M, NSE
90, 91, 92	30, 50, 100	15	C, M, NSE

Figure 15: Original experimental setup of COA[PC18].