COMPARISON OF GLOBAL AND LOCAL PARTICLE SWARM OPTIMIZATION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Nazrin Ferdousi

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2021

Fargo, North Dakota

North Dakota State University
Graduate School

Title

COMPARISON OF GLOBAL AND LOCAL PARTICLE SWARM OPTIMIZATION

By

Nazrin Ferdousi

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Saeed Salem

Dr. Maria de los Angeles Alfonseca-Cubero

Approved:

November 5, 2021 — Date

Dr. Simone Ludwig — Department Chair

ABSTRACT

Particle swarm optimization is a computational algorithm used to optimize a solution through sequential processing of particles using a specified estimation of quality. The algorithm is inspired by biological systems such as bird and insect swarms. This paper focuses on the comparison between Local PSO and Global PSO. We have utilized eight functions to provide benchmarks to compare the optimizations provided by the two optimization strategies. This resulted in findings that indicate that global optimizations tend to be more effective than local optimizations when comparing final costs. Our research indicates that an automated approach to particle swarm optimization will benefit from employing a range of benchmark functions and implementing both local and global optimizations. Analysis of the various particle topologies are discussed, and benchmark functions are selected and analyzed in regard to their final costs, as well as the overall particle topologies that they produce.

ACKNOWLEDGMENTS

I am very grateful to Dr. Simone Ludwig for accepting me as her advisee at the very last minute and saving me from the upcoming disaster. Being a pregnant person, I have been struggling harder both physically and mentally, but her constant support, guidance and motivation helped me to come this far to finish this paper. I would also like to thank Dr. Saeed Salem and Dr. Maria de los Angeles Alfaonseca-Cubero, for accepting my request for being part of the committee.

DEDICATION

To my mother Ms. Salma Huda and my father Md. Nurul Huda for influencing me with their energy for life, thirst for knowledge, constant support, and care in every single step of my life.

To my baby Nameer for giving strength from inside my womb during my complicated pregnancy, inspiring me to study and finish my graduation successfully, even when I was hospitalized twice during my postpartum period, and teaching me that life is hard but so very beautiful and there is so much to smile about.

To my husband Imran for being kind and considering.

To my best friends Aneela and Josh for their moral support and being beside me each time while I was facing any obstacles.

Last but not the least, to Allah SWT' for answering my prayers, giving me solidity, soundness, and strong willpower to graduate on time apart from being a sleep deprived mom overburdened with lots of responsibilities.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

1. INTRODUCTION

Particle swarm optimization (PSO) is important for the effective use of drones and unmanned aerial vehicles. To ensure that drones in a swarm do not crash into one another each, drone must know where the others are in a three-dimensional space. Drones also referred to as nodes or particles in this paper, can make use of PSO to both prevent collisions and optimize node to node communication speeds by finding the optimal positions that give the lowest communication latency costs between nodes. Evolutionary algorithms are employed to derive the optimal particle placement as the swarm moves through a three-dimensional space. The algorithm must properly account for the velocity, placement, and impacts of gravity, wind, and other such forces on the particles within the swarm. As particles of a swarm begin to optimize their positions, others emulate their movements and derive a global or local optimization with its placement in the swarm. The optimization strategies analyzed within this paper will be tested for both global and local optimization. PSO was developed by Eberhart and Kennedy in 1995 using a population of particles and a stochastic optimization technique named after the behavior of flocks of birds [1]. PSO was designed to incorporate swarming behaviors seen in the wild and displayed in the murmuration of flocks of birds [1]. These behaviors allow large swarms of birds to fly closely together without the risk of collisions. Similar behaviors are seen in swarming insects. Eberhart and Kennedy used to simulate these behaviors and the concept of PSO was the result of their research. Today there are many optimization problems that can be applied to using PSO and we will discuss several of them throughout this paper.

Swarm intelligence relies on PSO to allow man made devices to replicate natural swarming behaviors. With the increasing usage of unmanned arial vehicles both in the military and civilian sectors, the roles of swarm intelligence and optimization of these systems continues to become a

growing field of research. Not only does optimization allow for faster response times, but it can also be used to detect the locations of attackers. For example, a swarm coming under fire can triangulate the exact location of an attacker more effectively if each node is aware of its partners location and status [2]. PSO can also be utilized in predicting the future location of enemy drones, allowing for more effective targeting of enemy units [2]. In PSO each node is normally referred to as a particle and functions are utilized to find the optimal results for the particle's location in regard to costs. The main benefits of the PSO model is its fast convergence speeds. PSO is evaluated on benchmark functions such as the Rastrigin function, Sphere function, Ackley function, Rosenbrock function and so on. All particles' behavior in a swarm are considered to be affected by each other's particle behavior within the swarm. In real world applications there are other external inputs that can also have an effect on the swarm, for example, the location of a power wire, tree branch, bird or other physical object or wind speeds. This adds an additional level of complexity to the development of effective PSO. In this paper, we will look at two PSO variants, the Local and the Global versions.

1.1. Optimization

Optimization is a process used to locate and identify the best solutions for a specific set of conditions to determine the optimal solution. PSO is a heuristic and metaheuristic optimization technique based on those observed in nature [3]. A starling murmuration is an example of such natural optimization technique. Starling murmuration refer to the phenomenon that results when thousands of starlings fly in swooping, intricately coordinated patterns [3]. The optimization shown by the ingrained behaviors ensures that none of the birds ever collide with others in the group. Such optimization techniques can be used by humans in the case of swarms of unmanned arial vehicles. China, Russia and other nations have conducted experiments using UAVs

(Unmanned Air Vehicles) to create massive, coordinated light shows and the PSO technique could be employed to optimize such display, creating amazing graphical shows while ensuring that particles in this case drones do not interfere with one another [4].

1.2. Optimization Problems

The function to be optimized is referred to as the objective function also known as the fitness function. The variables required by the function are its input values. Input variables may be constrained by simple bounds or complex constraints. The search space is the set of all possible positions within the constraint bounds. The neighborhood is the subset of the search space that contains a given position. The objective value is the result obtained by the objective function. All together the objective function, its corresponding search space and its constraints all represent an optimization problem.

The current paper will start with the basics of PSO and how it works with regards to Global Best PSO and Local Best PSO in Chapter 3. It will then go on to describe the various PSO variants that will be discussed as well as their implementation details. We will then analyze the experiments and derive some conclusions based on the experiment results in Chapter 4 and 5, respectively.

2. RELATED WORK

There is an extensive collection of work focused on the concept of PSO. The first major work relating to PSO was published by Eberhart, Kennedy and Shi in 1995 [1]. This work focused on replication social behaviors observed in bird swarms. There are many variations of the PSO algorithm that are being evaluated using various benchmark functions such as Sphere, Alpine, Rosenbrock, and Rastrigin. There are some similarities between PSO and Genetic Algorithms although the methods the algorithms use to traverse the search space are fundamentally unique.

2.1. Optimization Techniques

Many optimization techniques have been developed yet none are perfect for every use. Optimization techniques fall into a few basic categories. The first being deterministic techniques such as: algebraic techniques [5], branch-and-bound [6] and interval optimization techniques [7]. Next, we have those that fall into the category of stochastic techniques these include, Monte Carlo sampling [8], parallel tempering [9], simulated annealing [10], and stochastic tunneling [11]. There are also several heuristic and meta-heuristic optimization techniques, including ant colony optimization [12], cuckoo search [13], evolutionary strategies [14], genetic algorithm [15], memetic algorithms [16], and of course PSO [1]. Wolpert and Macready introduced the "No free lunch" theorem based on research conducted by Wolpert on machine learning [17]. The "No free lunch" theorem is an impossibility theorem that states that depending upon the nature of the problem there can be no universal approach that will always outperform the others. The theorem is as follows: if there exists an optimization technique A that outperforms optimization technique B on a given optimization problem X there exists an optimization problem Y in which optimization technique B will outperform optimization technique A [17]. This theorem enforces the idea that multiple techniques are needed to achieve optimal optimization of a wide range of problems.

2.1.1. Local Best Particle Swarm Optimization

Local best PSO uses the ring network topology and small neighborhoods are defined for each particle. Information is exchanged between particles as they interact with each other's neighborhoods. This is local knowledge of the environment and is used to calculate the best local position based on the particle's neighborhood, neighboring particles and velocities.

Classical PSO formula

$$x(i)(n + 1) = x(i)(n) + v(i)(n + 1), n = 0, 1, 2, ..., N – 1 \quad (1)$$

Local best velocity formula

$$vij(n + 1) = vij(n + 1) + c1r1(n) [yij(n) – xij(n)] + c2r2(n) [y (n) – xij(n)] \quad (2)$$

Using the above formulas local best PSO generates Local best values for each particle within the swarm. Velocity is calculated using the second formula.

2.1.2. Global Best Particle Swarm Optimization

Global best PSO is a standard variation of PSO where the swarm particles are initialized randomly with random velocities within the search space. The objective function or fitness function is then implemented with the best objective fitness values being those already assigned to the particles. With the global best being that of the best particles position and values within the entire swarm.

Global best function

$$v^i(n + 1) = wv^i(n) + C_1r_1{}^i(n)[x_p{}^i(n) – xi(n)] + (n) [xg(n) – xi(n) \quad (3)$$

where n = 0,1,2,…,N-1.

The Classical PSO formula function then works on moving the particles and the fitness values are evaluated once again and updated with their personal best values. The global best position is then updated and replaced by any new best value.

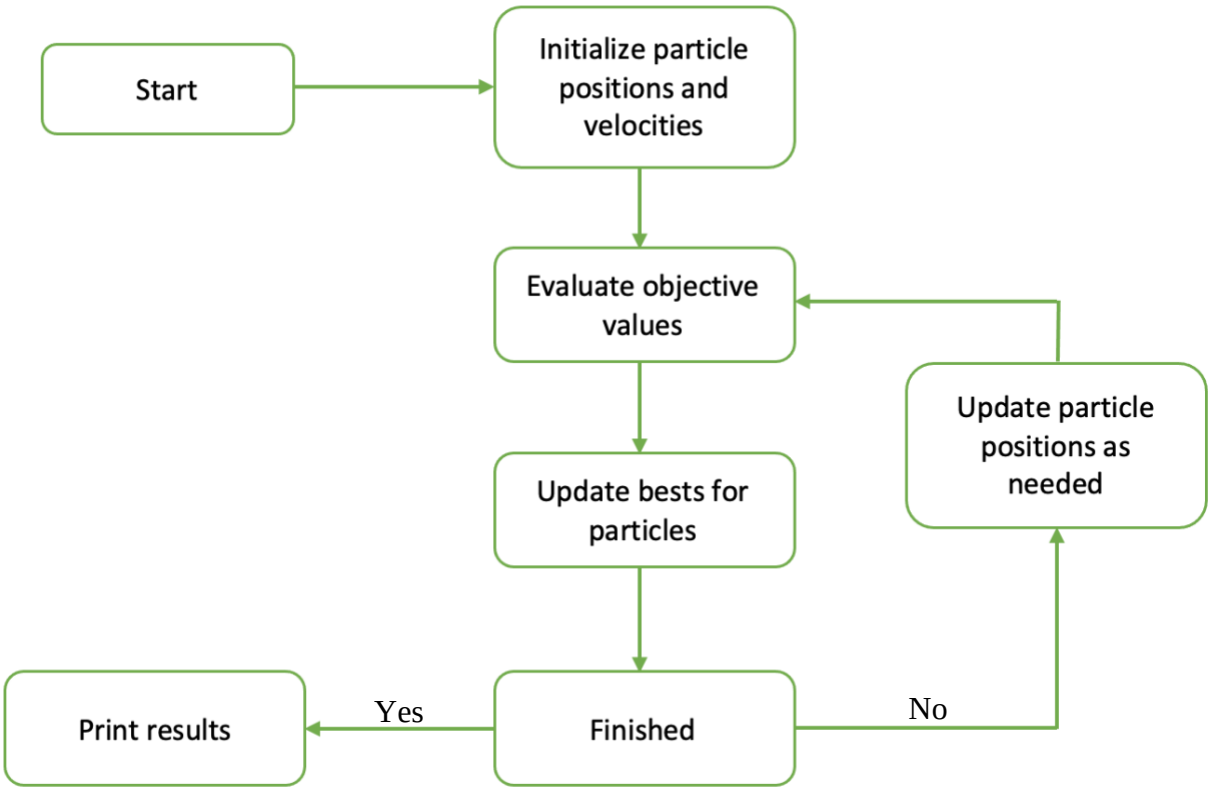

Figure 1. Flow chart of the PSO process.

Figure 1 shows the flowchart representing the logic of the PSO algorithm. Particle positions are initialized with random positions and velocities. The evaluation phase uses iteration steps to check the fitness of all particles then the update step logs the global and or local best. If the process is finished and a suitable solution has been found the result will be printed else the particles' locations are updated and the loop will start once more at the evaluation phase with the particles moved into new positions.

3. PARTICLE SWARM OPTIMIZATION

The PSO variants (global and local) will be used with a set of benchmark functions to find the optimal solution for each benchmark function. This section will discuss the variants used for the benchmark functions and break down their benefits and use cases. As stated earlier the "No free lunch" theorem states that there will never be a single algorithm that best optimizes every case. Thus, we will explore what use cases are best suited for each PSO variant in terms of optimality.

3.1. Adaptive Particle Swarm Optimization (APSO)

Adaptive particle swarm optimization (APSO) uses adaptive parameters and a learning strategy based on the "evolutionary state estimation" (ESE) resulting in what is called the "elitist learning strategy" (ELS). The value of the evolutionary state is developed using an evolutionary factor and its relative particle fitness within the population. Each generation goes through a classification process to evaluate the fitness of each particle and its attributes. Adaptive control strategies are useful in allowing the system to evolve and select the optimal positions and vectors using ELS to control the swarm's evolution.

The equations used in APSO define a fuzzy classification method. Where f is the evolutionary factor and n is the population size for the equations found in Formulas (4) and (5). Within the equation, d represents the dimension.

Step 1: Calculate the mean distance between particle 1 and all other particles using the equation in Formula (4).

$$d_i = \frac{1}{n-1} \sum_{j=1, j \neq i}^{n} \sqrt{\sum_{k=1}^{d} (x_i{}^k - x_j{}^k)^2} \tag{4}$$

Step 2: After the mean distance has been found and assigned to d_i for each particle then all values will be compared to locate the maximum and minimum distance d_{max} and d_{min}. The global best will then be denoted as d_g and the evolutionary factor f is defined using the equation in Formula (5), which is initially set to the default value of 1 when the algorithm starts.

$$f = \frac{d_g - d_{min}}{d_{max} - d_{min}} \in [0,1] \tag{5}$$

Step 3: Once the evolutionary factor f has been calculated the evolutionary state is divided into four substates: convergence, exploitation, exploration, and jump-out states. Inertia weight is used to balance out global and local search abilities and should generally decrease in a linear fashion at a rate of 0.9 to 0.4 with each generation. Formula (6) shows the equation used to adjust inertial weight, and Formula (7) shows the equation used for acceleration coefficients and particle position adjustments.

$$w(f) = \frac{1}{1 + 1.5e^{-2.6f}} \quad \in [0,1] \tag{6}$$

$$c_i\,(n+1) = c_i\,(n) \pm \Delta\,, where\ i = 1,2 \tag{7}$$

where Δ represents a value within the range 0.05 to 0.1 which is randomly generated in a uniform fashion to ensure against stagnation of the particles.

Step 4: Select the best position denoted p_d. The formula used to find the best position can be found in Formula (8).

$$p_d = p_d + (x^d{}_{max} - x^d{}_{min}) Gaussian(\mu, \sigma^2) \tag{8}$$

The Gaussian distribution in Formula (8) uses a mean μ set to 0 with a time varying standard deviation where $\sigma_{min} = 0.1$ and $\sigma_{max} = 1.0$ with g being the local best value and G being the global best value. The equation used to derive σ can be seen in Formula (9).

$$\sigma = \sigma_{max} - (\sigma_{max} - \sigma_{min})(\frac{g}{G}) \quad (9)$$

Table 1. Strategies used to obtain values for c1 and c2.

Strategies	**C1**	**C2**	**States**
Option 1	Increase C1	Decrease C2	Exploration
Option 2	Small increase in C1	Small decrease in C2	Exploitation
Option 3	Small increase in C1	Small increase in C2	Convergence
Option 4	Decrease C1	Increase C2	Jump-out

Table 1 shows the particle manipulation strategies that can be used to manipulate the particles to better achieve optimal particle placement and avoid particles from being trapped in sub optimal local maxima states.

3.2. Algorithm Aspects

3.2.1. Particle Initialization

Particles within a neighborhood are initialized initially at random locations with random vectors. Upon each iteration formulas can be used to adjust the particle's locations with each successive iteration producing an evolving swarm topology.

3.2.2. Fitness Evaluation

Fitness functions are used to evaluate the fitness of a particle and update its corresponding fitness score. Depending upon global or local evaluation these functions can range from simple to complex and constraints may be utilized to cull out less fit positions to direct particles to more optimal positions. In other cases, as seen with simulated annealing, particles may be allowed to go back to less optimal positions at times [10].

3.2.3. Stopping Conditions

The stop conditions are used to terminate the algorithm after a set number of iterations has been executed. Results are then printed, and any relevant information is graphed for future analysis or use. Stop conditions can be adjusted as needed to allow for a high degree of optimality or lower optimality as required by the specific use case requirements.

4. EXPERIMENTS AND RESULTS

All benchmarks were conducted using PySwarms and the python programming language using both local and global particle swarm optimization techniques on each benchmark function. The settings for the number of particles, dimensions, number of iterations and other input variables remained consistent for each function tested. Note that in our experiments we used three dimensions for all of our benchmarks.

The program runs both Local Best and Global Best PSO using the benchmark functions listed in this chapter. Input variables are used consistently to allow for equal comparison between the differing benchmark functions. Inputs include number of iterations, global increment, particle increment, particle count, and dimensions. The results were then obtained using the pyswarms.single.global_best module and pyswarms.single.local_best module [18].

4.1. Pyswarms Module Functionality

Pyswarms uses two distinct modules and a few different parameters to provide functional options. Those functionalities and parameters are discussed below in detail which will be performing optimizations of objective functions using the global-best optimizer in pyswarms.single.GBestPSO and the local-best optimizer in pyswarms.single.LBestPSO. This aims to demonstrate the basic capabilities of the library when applied to benchmark problems [18].

4.1.1. Pyswarms.single.global_best Module Functionality

This function implements a Global-best Particle Swarm Optimization (gbest PSO) algorithm. The function takes a set of candidate solutions and attempts to find the best possible solution by using a position-velocity update method. The function uses a star topology where each particle is drawn to the best performing particle. The position update is defined using the formula shown in Formula (10) where the position at a given timestep t is updated using a computed

velocity calculated at t + 1. The formula used to calculate the velocity is shown in Formula (11). The velocity formula defines c1 and c2 as cognitive and social parameters that control the behavior of a given particle given two choices: (1) follow its personal best position, (2) follow the swarm's global best position. The result of this choice determines if the swarm is explorative or exploitative with the variable w controlling the inertia of the swarm's movement.

$$xi(t+1) = xi(t) + vi(t+i) \tag{10}$$

$$vij(t+1) = w * vij(t) + c1r1j(t)[yij(t) - xij(t)] + c2r2j(t)\left[y^{j(t)} - xij(t)\right] \tag{11}$$

The global best PSO algorithm was adapted from the works of Kennedy and Eberhart in Particle Swarm Optimization [1]. The algorithm used in global best PSO can be seen represented using pseudo code in Figure 2.

```
Create and initialize swarm
While stop condition = false
        For each particle in swarm do
                If p_k^i < p^i_best then
                        P^i_best = p^i_k
                Set global best
                If p^i_k < p_gbest then
                        p_gbest = p^i_k
        for each particle do
                update velocity
                update position
        Check and update stop condition
```

Figure 2. Global best PSO algorithm.

4.1.2. Pyswarms.single.local_best Module Functionality

The Local-best Particle Swarm Optimization (lbest PSO) algorithm is similar to the global-best PSO algorithm. The Local best algorithm takes a set of candidate solutions, then attempts to find the best solution using a position-velocity update method. The local best PSO algorithm uses a ring topology, this results in the particles being attracted to its corresponding neighborhood.

The position update formula can be seen in Formula (12) with t being the timestep being updated using the computed velocity at t + 1. The formula used to update the computed velocity at t + 1 being the same as that shown in Formula (10).

$$xi(t+1) = xi(t) + vi(t+1) \tag{12}$$

Local PSO differs from Global PSO in that a given particle does not compare itself to the overall swarm's performance but instead looks at the performance of its closest neighbors and makes comparisons with them for optimality. Local best PSO generally takes much more time to converge when compared to Global best PSO but offers better explorative features. Pyswarm's implementation of this algorithm uses a k-D tree imported from SciPy to select neighbors and distance is computed using either the L1 or L2 distance. The nearest neighbors are then selected from the resulting k-D tree and the tree is recomputed with every iteration. The local best PSO algorithm was adapted from the works of Kennedy and Eberhart in Particle Swarm Optimization [1].

4.1.3. Parameters

- Options
- Number of iterations
- Number of particles
- Bounds

The options used were [c1: 0.5, c2: 0.3, w: 0.9] for global optimizations with c1 being the cognitive parameter, c2 being the social parameter and w being the inertia parameter. 1,000 iterations were used in each test with 50 particles. Bounds were set depending on the specific function in use ranging from 5.12 to 10 using the negatives as lower bounds.

4.2. Experiment Goals

The goals of the experiments included in this paper is to develop effective methods to control a swarm's topology to reduce the communication costs between individual nodes. Thus, we are using a series of benchmark functions to find costs and particle positions in both local and global PSO. When run together as a single program the optimal solution can be utilized to coordinate particle movement and maintain the most efficient topology to reduce the communication costs. This can also be combined with the k nearest neighbors' algorithm to create a digital evolving record of particle placement. The various fitness functions used each have their own best-case topologies and if combined with machine learning the system could predict what fitness function to utilize based on the shape of the swarm's topology in real time. As of now each function will be run sequentially and the best optimization can be analyzed individually to determine the best topology and particle placement for optimization.

4.3. Ackley

The Ackley test function can have several local optima located at regular intervals. This fitness function will produce a single global optima located at the point 0, 0, 0. The function is shown in Formula (13).

$$f(x) = -ae^{-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i{}^2}} - e^{\frac{1}{d}\sum_{i=1}^{d} cos\,(cx_i)} + a + e^1 \tag{13}$$

Figure 3 shows the optimal topology of the Ackley fitness function. The Ackley function is a concave function with the optimal point being the center point. The global minima is obtained by f(0,0)=0, with a domain of -5 <= x, y <= 5.

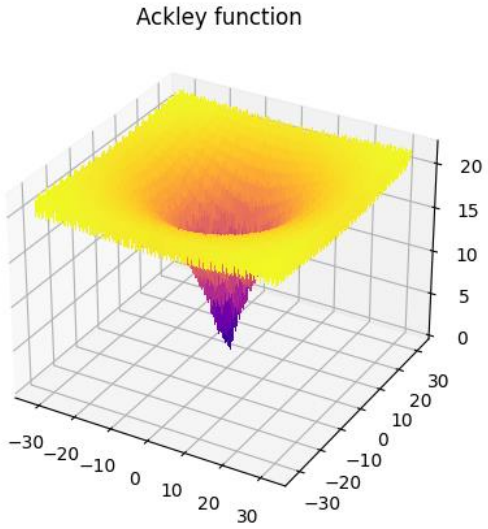

Figure 3. Topology of the Ackley function.

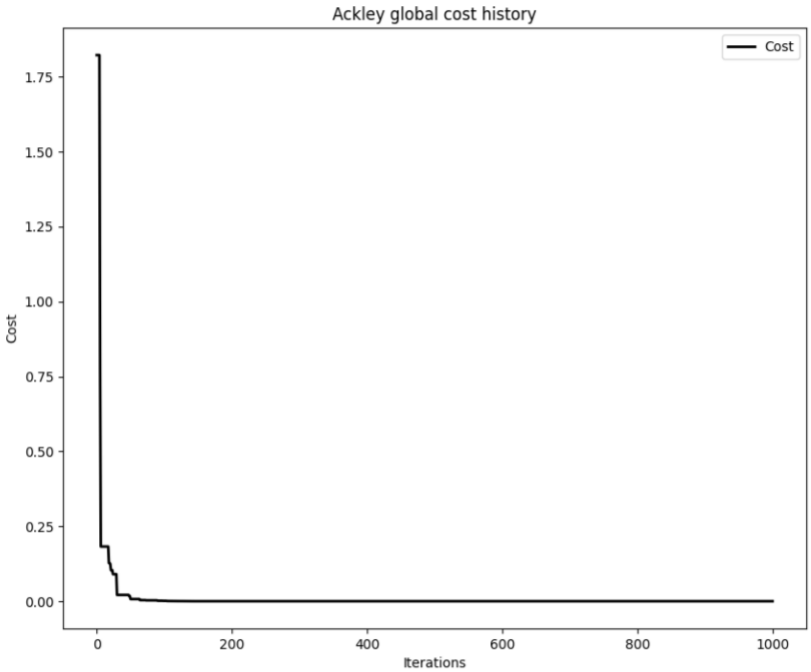

Figure 4. Global best cost history for the Ackley function.

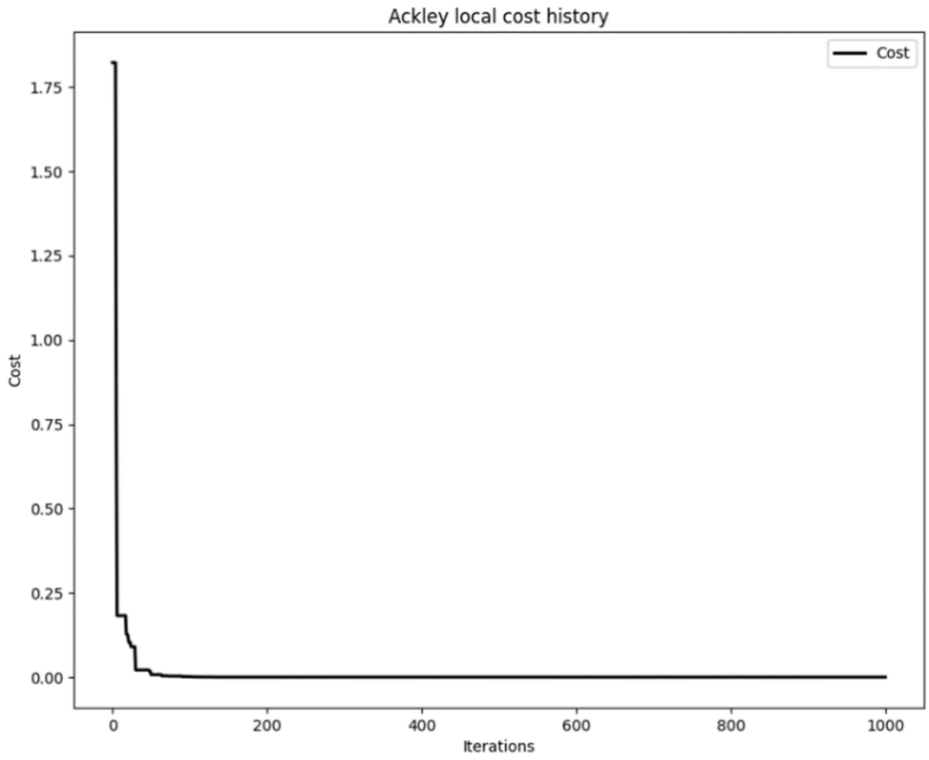

Figure 5. Local best cost history for the Ackley function.

Figure 4 shows the number of iterations needed to find a suitable cost. A suitable cost is one that is close to 0. As we can see in Figure 5 an optimal solution can be found in under 50 iterations. Figure 5 shows a very similar drop in cost from 1.37 to almost 0 around the same number of iterations.

Table 2. Ackley global fitness function results.

Global best cost	4.440892098500626e-16
Global best position	[1.45519618e-17, 4.21114928e-16, -1.81346765e-16]

Table 3. Ackley local fitness function results.

Local best cost	0.05203503379902985
Local best position	[0.00875255, 0.01662873, 0.00552959]
Local best position options	'c1': 4.002561740255713, 'c2': 6.132272490604348, 'w': 4.821831298950279, 'k': 13, 'p': 1

Table 2 contains data relating to the best costs of global optimization along with the corresponding best positions associated with the local optimization in Table 3. As we can see when comparing the two the global optimization produced the lowest cost at 4.440892098500626e-16 compared to 0.05203503379902985. The changes needed to the options in the local best optimization are shown in the table as well.

4.4. Levi

The Levi fitness function is best for ramp like topologies as we can see in Figure 6. The Levi function is as follows:

$$f(x,y) = sin^2 3\pi + (x-1)^2(1+sin^2 3\pi y) + (y-1)^2(1+sin^2 2\pi y) \qquad (14)$$

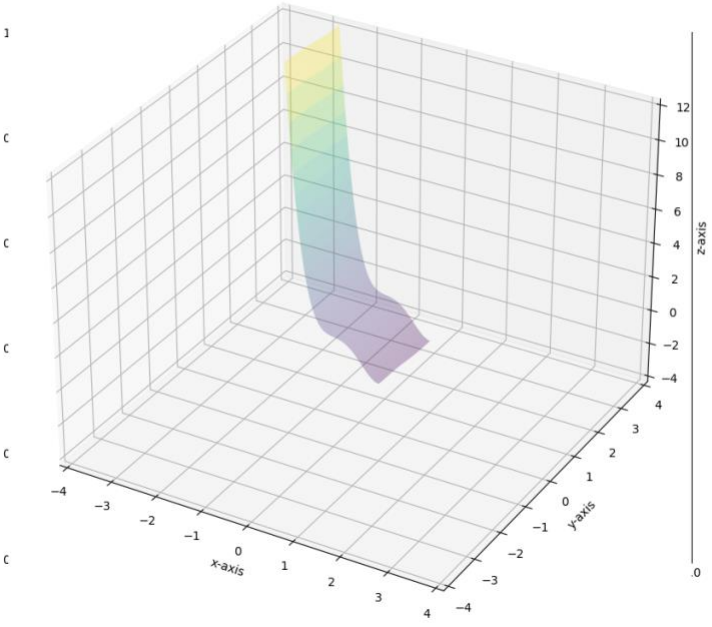

Figure 6. Topology of the Levi function.

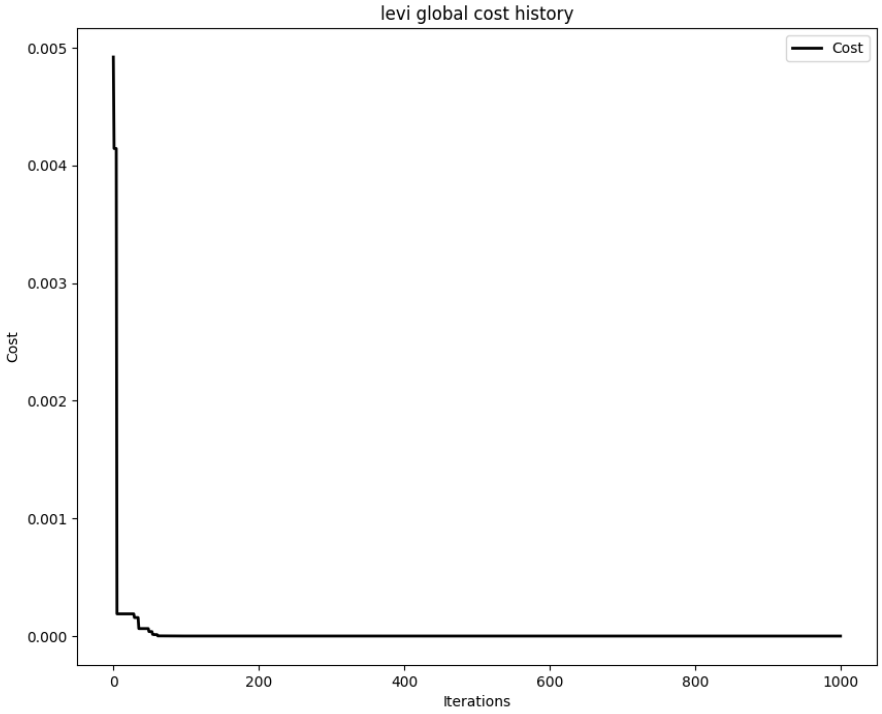

Figure 7. Global best cost history for the Levi function.

Optimal solutions are found for the global best position using the Levi function very quickly as shown in the cost history graph seen in Figure 7. When we look at the local best cost history shown in Figure 8, we see very little difference between the two. We will see more change with some of the other fitness functions later in this section.

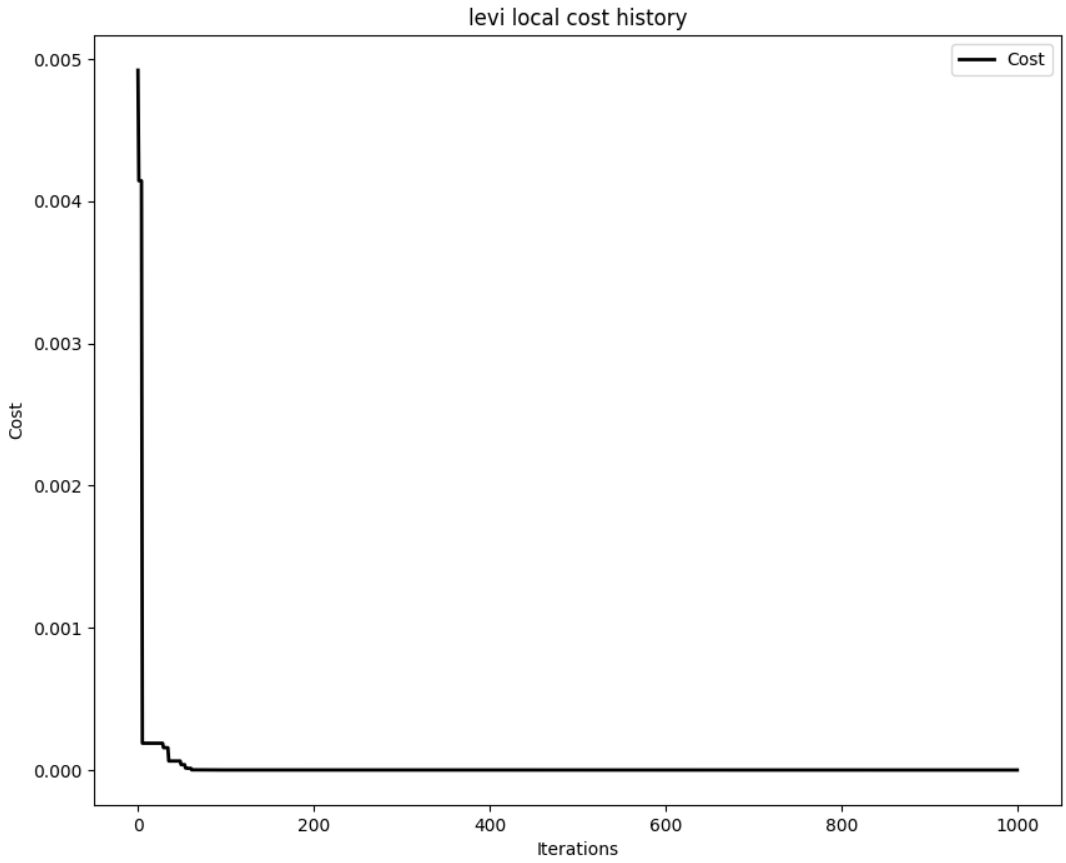

Figure 8. Local best cost history for the Levi function.

Table 4. Levi global fitness function results.

Global best cost	1.4997597826618576e-32
Global best position	[1, 1, 0]

Table 5. Levi local fitness function results.

Local best cost	2.9034752985917007e-05
Local best position	[0.99997622, 0.97846058, 0]
Local best position options	{'c1': 2.1724360683338735, 'c2': 9.62486041013009, 'w': 4.3099398943224365, 'k': 12, 'p': 1}

Table 4 shows the results obtained for global optimization using the Levi function and Table 5 shows the corresponding results of local optimization using the Levi fitness function. As we see the global optimization produces a lower cost than local optimization.

4.5. Rastrigin

The Rastrigin function is based on the De Jong function with the addition of Cosine modulation to allow the production of frequent local minima. The function is defined using the following formula:

$$f(x) = \sum_{i=1}^{d} [({x_i}^2 - 10\,cos(2\pi x_i) + 10] \tag{15}$$

where d is the dimension and x is a d-dimensional row vector or 1 by d matrix. An orthogonal matrix is also utilized for coordinate rotation to make the function non-separable. The number of local optima grows exponentially as the dimensionality is increased. The test area is restricted to a hypercube starting at the limit -5.12 to 5.12 with its global minima f(x) = 0 found within that range. As we see in Figure 10 the Rastrigin function can take more time compared to some of the other fitness functions; this is due to it having many local maxima and local minima.

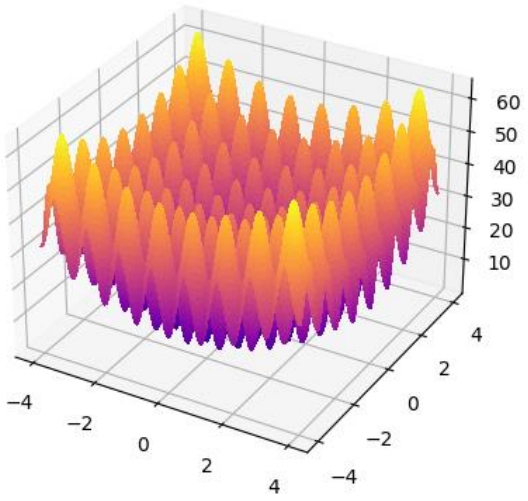

Figure 9. Topology of the Rastrigin function.

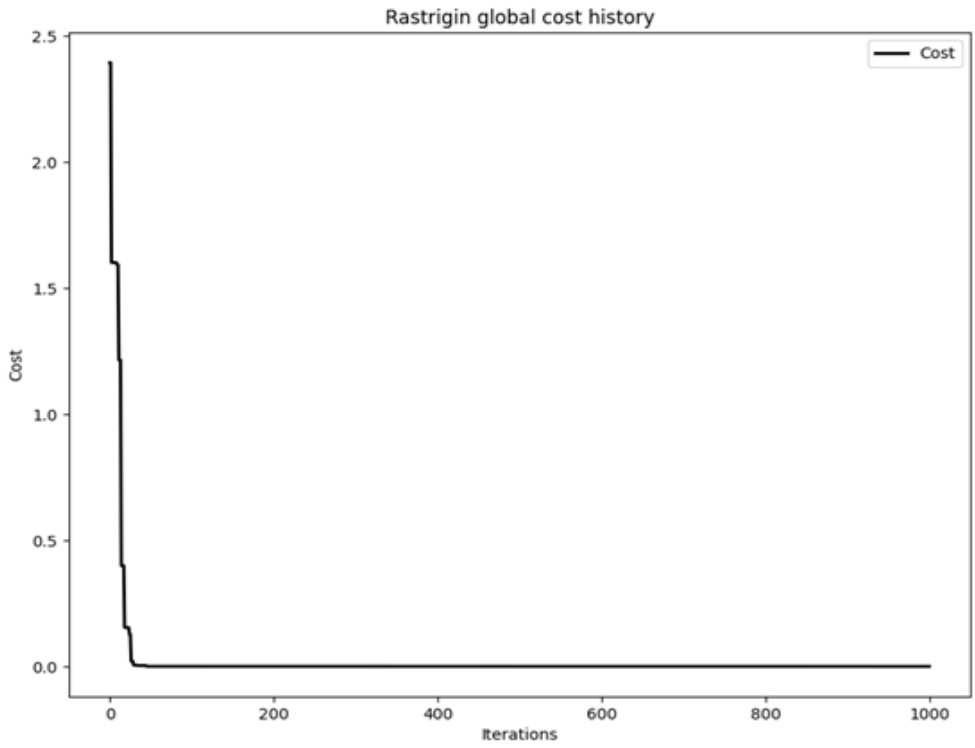

Figure 10. Global best cost history for the Rastrigin function.

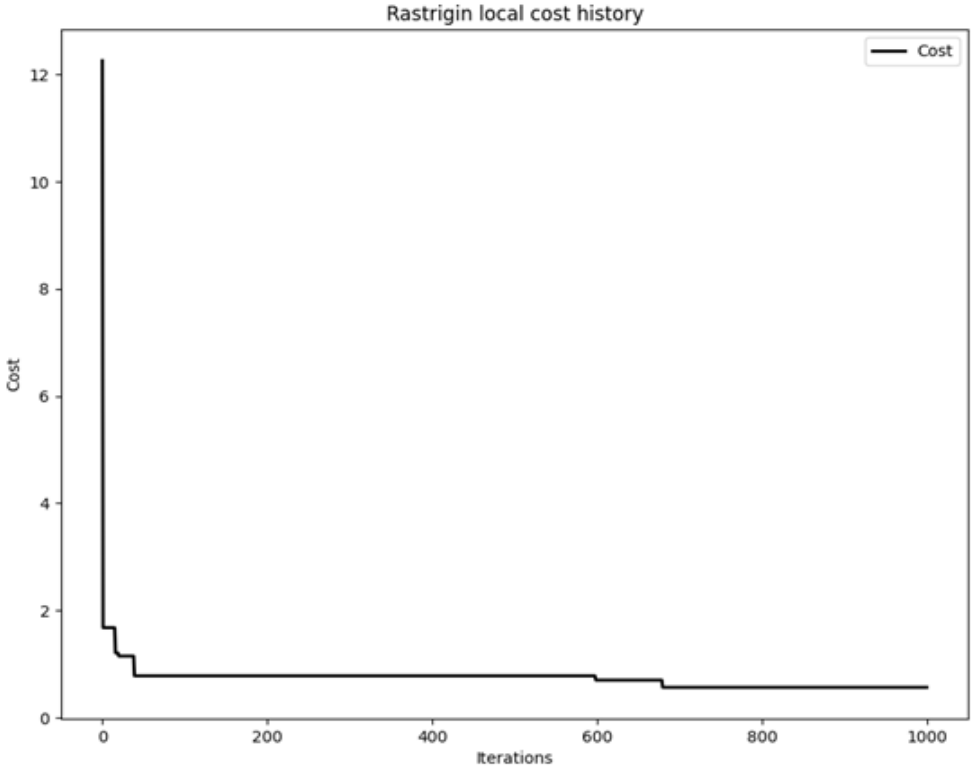

Figure 11. Local best cost history for the Rastrigin function.

Rastrigin produced a best cost of 0.00026 with a best position of 0, 0, 0 as seen in Table 6 for the global optimization where the local optimization produced a cost of 0.9180498767704037 as seen in Table 7.

Table 6. Rastrigin global fitness function results.

Global best cost	0.00026
Global best position	[0, 0, 0]

Table 7. Rastrigin local fitness function results.

Local best cost	0.9180498767704037
Local best position	[0.02692257, 0.01383459, 0.06131146]
Local best position options	{'c1': 1.1452949295076018, 'c2': 1.086624142067919, 'w': 1.0739533224992848, 'k': 1, 'p': 1}

4.6. Rosenbrock

The Rosenbrock function is considered a non-convex function, developed by Rosenbrock in 1960. The global minimum is inside a long, narrow, parabolic shaped flat valley. The function is defined as follows:

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - {x_i}^2)^2 + (1 - x_i)^2] \tag{16}$$

where $x = (x_1, \ldots, x_n)$ and $n = 3$.

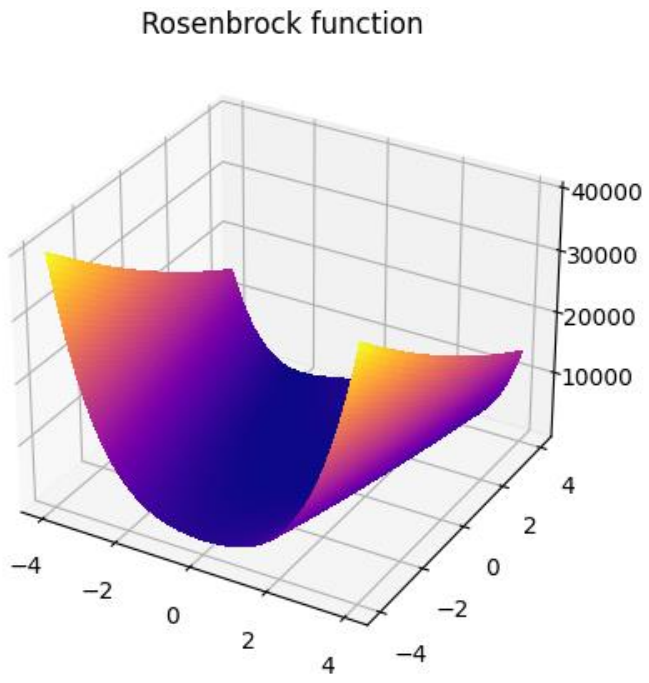

Figure 12. Topology of the Rosenbrock function.

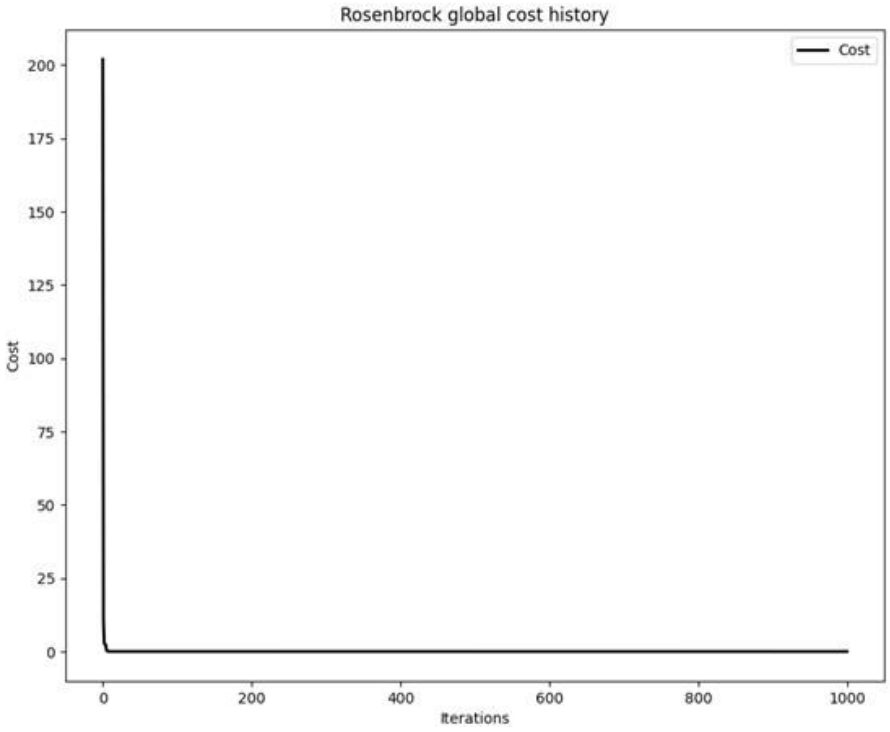

Figure 13. Global best cost history for the Rosenbrock function.

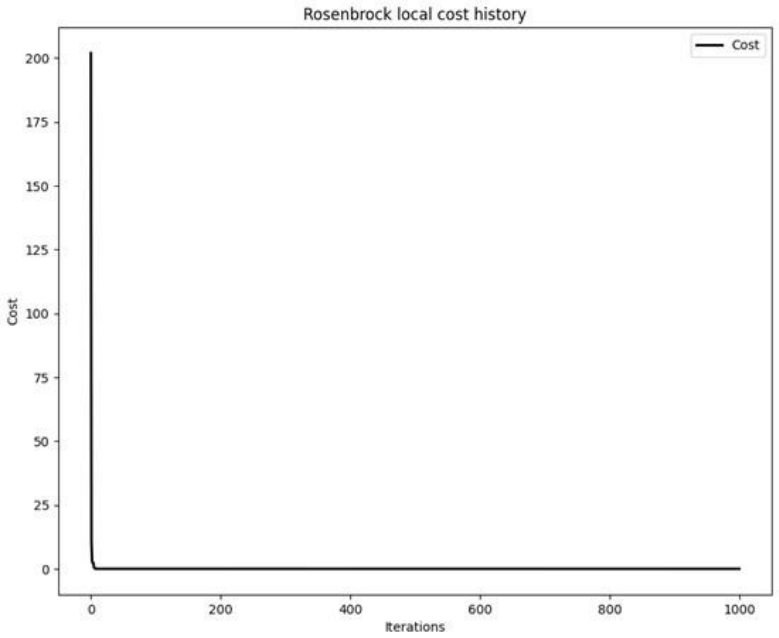

Figure 14. Local best cost history for the Rosenbrock function.

As we see in Figure 14 the Rosenbrock function produces a local best at around 80 iterations with a very high initial cost of 50. In Figure 12 we see the overall topology of the function with the optimal position being as close to the point (0, 0, 0) as possible. As we see in Figure 13 depending upon the particles starting positions it may take many iterations for the particles to converge. Table 8 contains the resulting cost of global optimization and Table 9 contains the results of local optimization using the Rosenbrock fitness function.

Table 8. Rosenbrock global fitness function results.

Global best cost	4.3651867427948036e-14
Global best position	[1.00000012, 1.00000021, 0]

Table 9. Rosenbrock local fitness function results.

Local best cost	0.04275490910466276
Local best position	[0.91951961 0.83882813 0.69599832]
Local best position options	{'c1': 2.6197841082614404, 'c2': 8.183208143758625, 'w': 2.4488263690238306, 'k': 13, 'p': 1}

4.7. Schaffer2

The Schaffer2 fitness function is normally bounded with a domain of -100 to 100. With a global minima at f(x) = 0 at x = (0, 0). The functions definition is as follows:

$$f(x) = 0.5 + \frac{sin^2({x_1}^2 - {x_2}^2) - 0.5}{[1 + 0.001({x_1}^2 - {x_2}^2]^2} \quad (17)$$

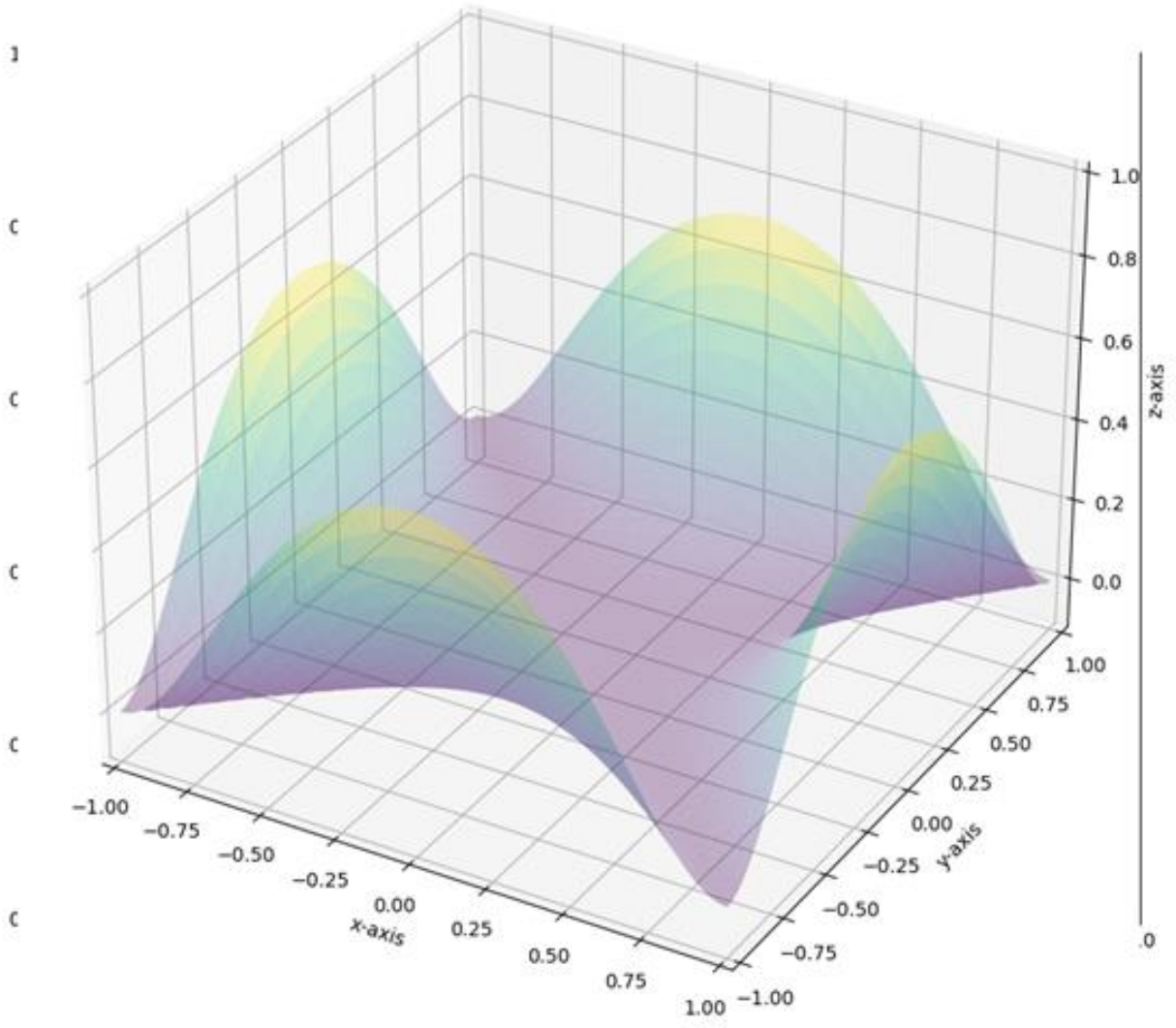

Figure 15. Topology of the Schaffer2 function.

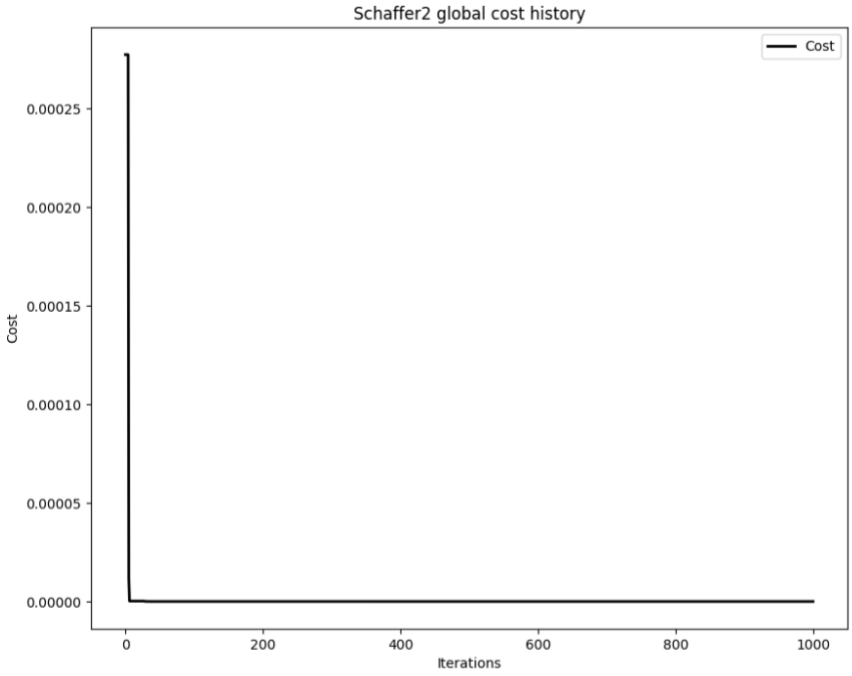

Figure 16. Global best cost history for the Scaffer2 function.

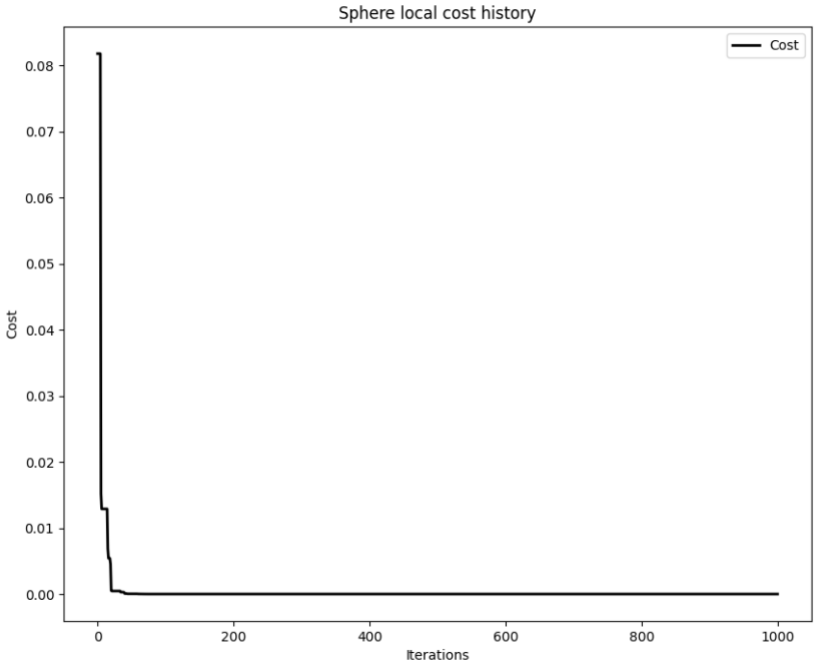

Figure 17. Local best cost history for the Scaffer2 function.

As we see in Figure 16 the initial cost is not terribly high for the Schaffer2 fitness function, and it tends to find an optimal solution within a few iterations. Figure 17 contrasts as the local optimization takes longer to find a solution.

The topology as seen in Figure 15 prevents the existence of independent local minima but has 4 local maxima. As the algorithm progresses the particles are drawn to the global minima in the center quite quickly making it an efficient and effective function. Tables 10 and 11 contain the resulting costs of global optimization and local optimization, respectively.

Table 10. Schaffer2 global fitness function results.

Global best cost	0.0
Global best position	[3.90009005e-09 -3.20968894e-07, 3.54832001e]

Table 11. Schaffer2 local fitness function results.

Local best cost	4.7471594005754625e-07
Local best position	[0.01492195, 0.0158504, 0.01532846]
Local best position options	{'c1': 2.1483513747567646, 'c2': 8.170510960805752, 'w': 4.475328421750222, 'k': 14, 'p': 1}

4.8. Sphere

The Sphere function is very useful as many other topologies can be put into a spherical topology. The domain for this function is quite extensive: $-\infty \leq x_i \leq \infty$, $1 \leq i \leq n$.

The function definition is as follows:

$$f(x) = \sum_{i=1}^{n} {x_i}^2 \tag{18}$$

With a optimal solution at $f(x_1, \ldots, x_n) = 0$.

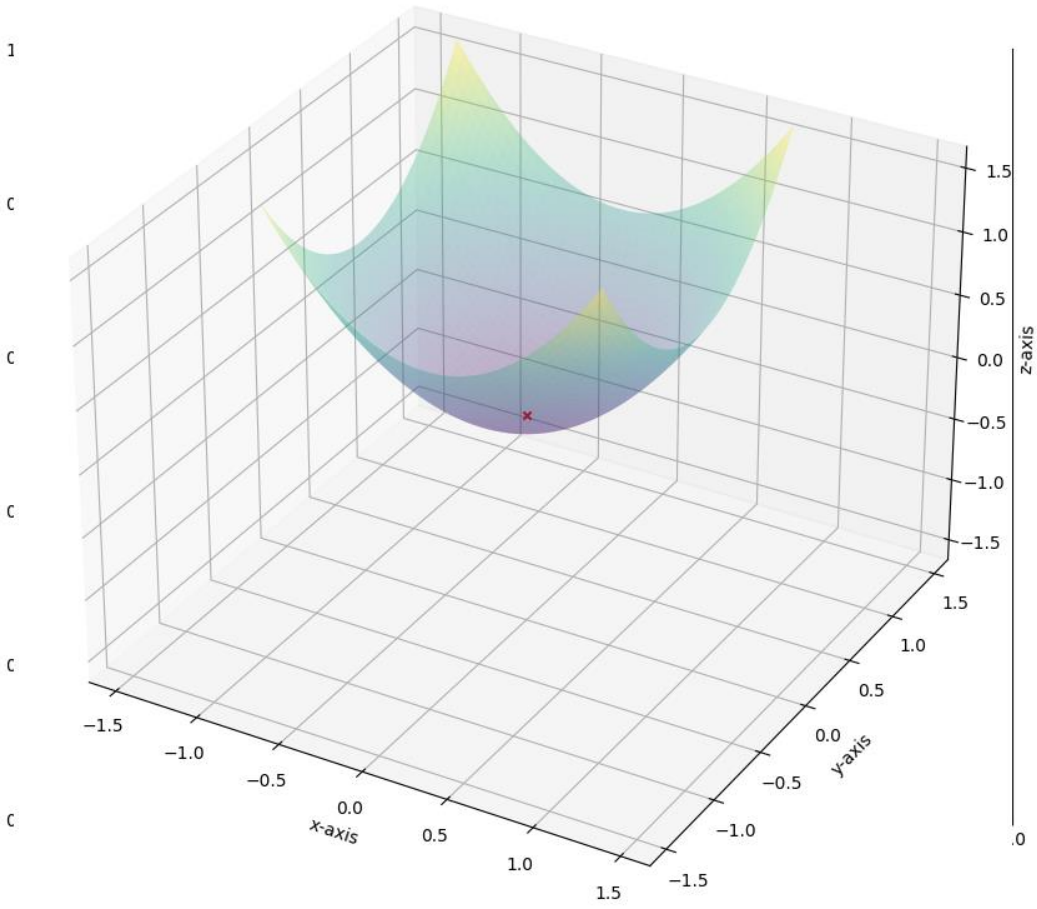

Figure 18. Topology of the Sphere function.

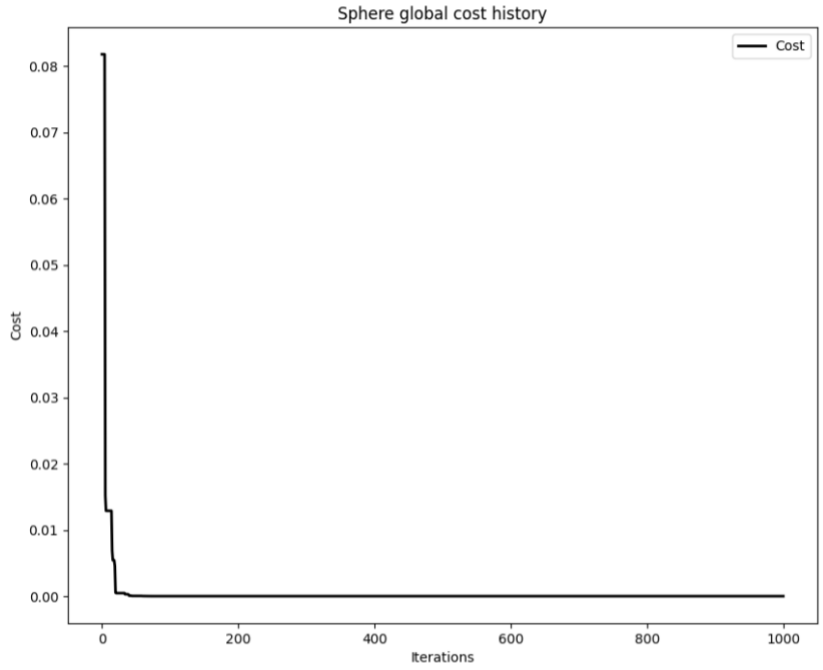

Figure 19. Global best cost history for the Sphere function.

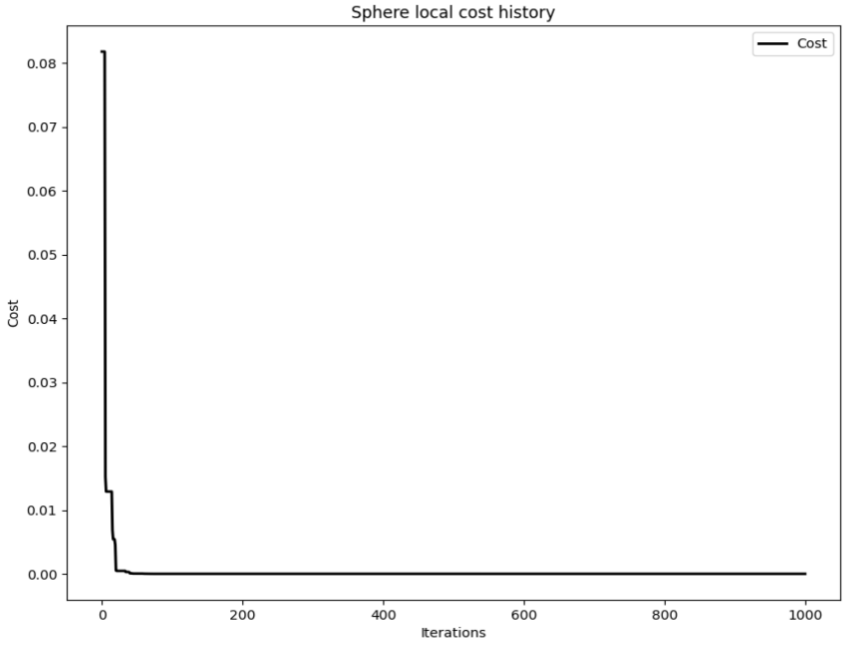

Figure 20. Local best cost history for the Sphere function.

As seen in Figures 19 and 20 the optimal solution is found very effectively using the Sphere function with no major difference in time taken between global and local optimizations. Figure 18 shows the basic topology of the function and its global optima at point (0, 0, 0).

The Sphere function is one of the most effective fitness functions tested and has a broad application range. Table 12 contains the functions results for global optimization which are obtained very quickly as seen in Figure 19. Table 13 displays the results obtained by local optimization using the Sphere function. There are some limitations on the use of this function as swarms will seldom completely match a spherical topology although due to the simplicity and speed of the algorithm it may be more suitable as a general-purpose fitness function than others especially considering that the domain for this function is far more inclusive than many of the other functions available. The question here would be if the execution speed is more important than a perfect solution as no single function can meet the needs of every possible case. The Sphere function can easily serve as a default fitness function.

Table 12. Sphere global fitness function results.

Global best cost	0.0
Global best position	[-1.27259802e-162, -2.48470447e-163, -2.93072783e-163]

Table 13. Sphere local fitness function results.

Local best cost	0.005153627760837311
Local best position	[0.02606911, 0.06472562, 0.01687078]
Local best position options	{'c1': 3.225031431243699, 'c2': 6.876527316376868, 'w': 4.805224838818529, 'k': 12, 'p': 1}

4.9. Styblinski / Tang

The Styblinski – Tang function is usually restricted to the domain: x_i in [-5, 5] for all i = 1, …, d where d is the dimension and in our case d = 3. The formula is as follows:

$$f(x) = \frac{\sum_{i=1}^{n} {x_i}^4 - 16{x_i}^2 + 5x_i}{2} \tag{19}$$

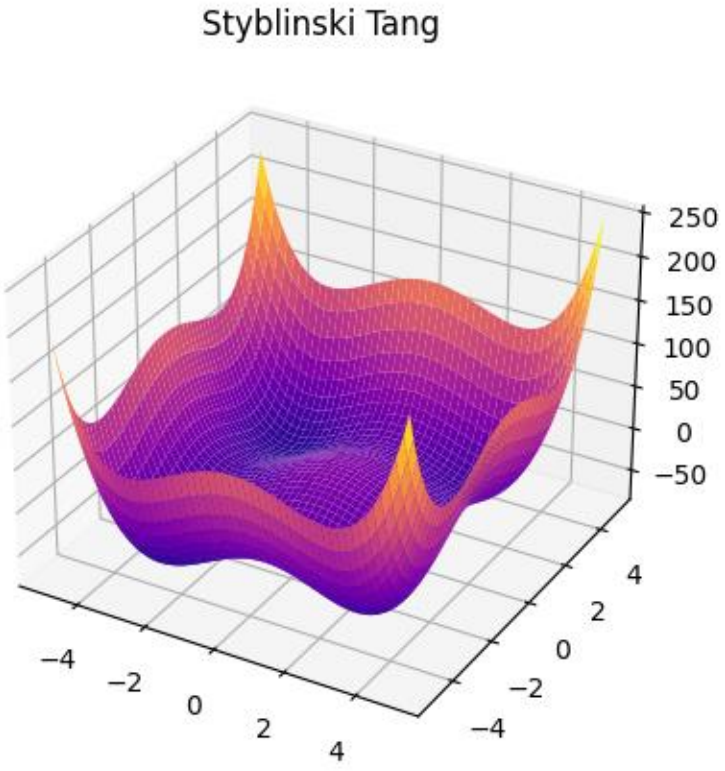

Figure 21. Topology of the Styblinski-Tang function.

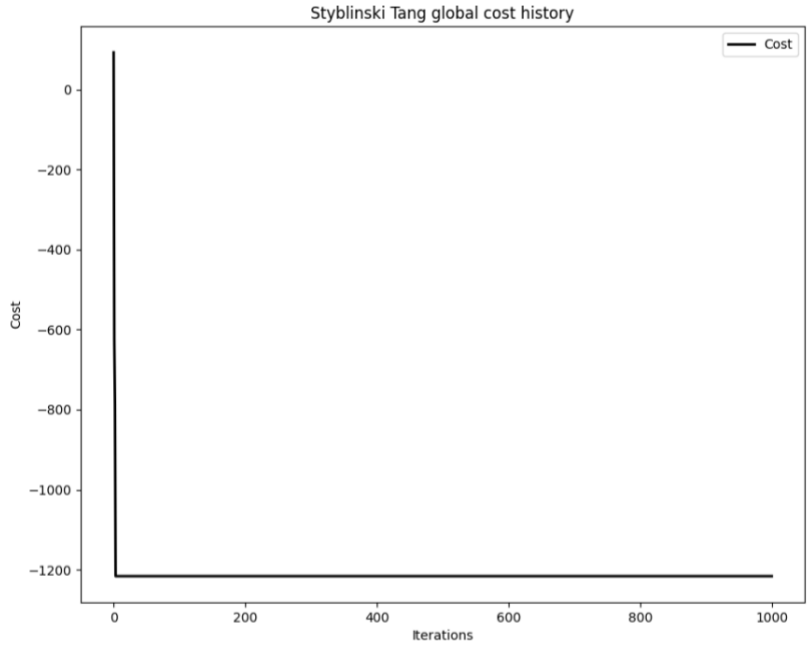

Figure 22. Global best cost history for the Styblinski Tang function.

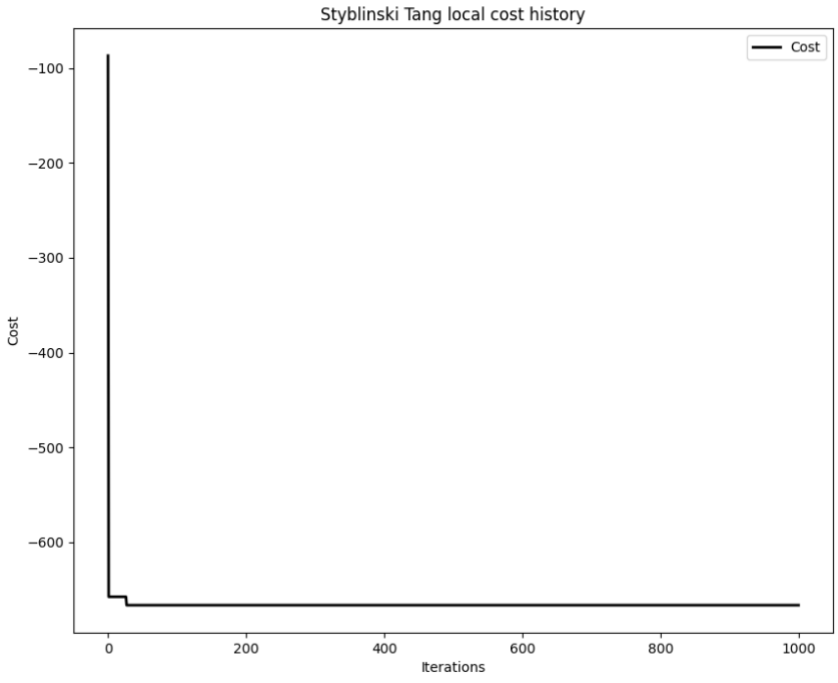

Figure 23. Local best cost history for the Styblinski Tang function.

As we see in Figure 22 this function can be exceptionally slower than some of the others we have tested and has limited application in swarm topologies. This function performs better in local best optimizations as seen when we compare Figures 22 and 23. Figure 21 shows the optimal topology of this test function. While this function is much less effective overall especially compared to the Sphere function there are times when the swarm's topology may be better served by this fitness function.

Table 14. Styblinski / Tang global fitness function results.

Global best cost	-1272.781987190529
Global best position	[-3.80601064, -0.08326784, -129.68240329]

Table 15. Styblinski / Tang local fitness function results.

Local best cost	-1156.6715579144793
Local best position	[2.15530099, 2.13629179, 122.60836240]
Local best position options	{'c1': 4.439324880921251, 'c2': 7.628149707706805, 'w': 4.5295616846535305, 'k': 11, 'p': 1}

The results seen in Tables 14 and 15 were derived using the same options input used in other local best benchmarks but in this case have produced unsatisfactory results. This could be due to the majority of particles being so far from one another making both local and global bests difficult to pinpoint.

4.10. Three Hump

This function also called the Three humped camel function oddly does not resemble a camel when graphed topologically. The recommended input domain of this function is (-5,5) with a global minima located at $f(x) = 0$ with $x = (0, 0)$. The function is defined as follows:

$$f(x) = 2{x_1}^2 - 1.5({x_1}^4) + \frac{{x_1}^6}{6} + x_1 x_2 + {x_2}^2 \tag{20}$$

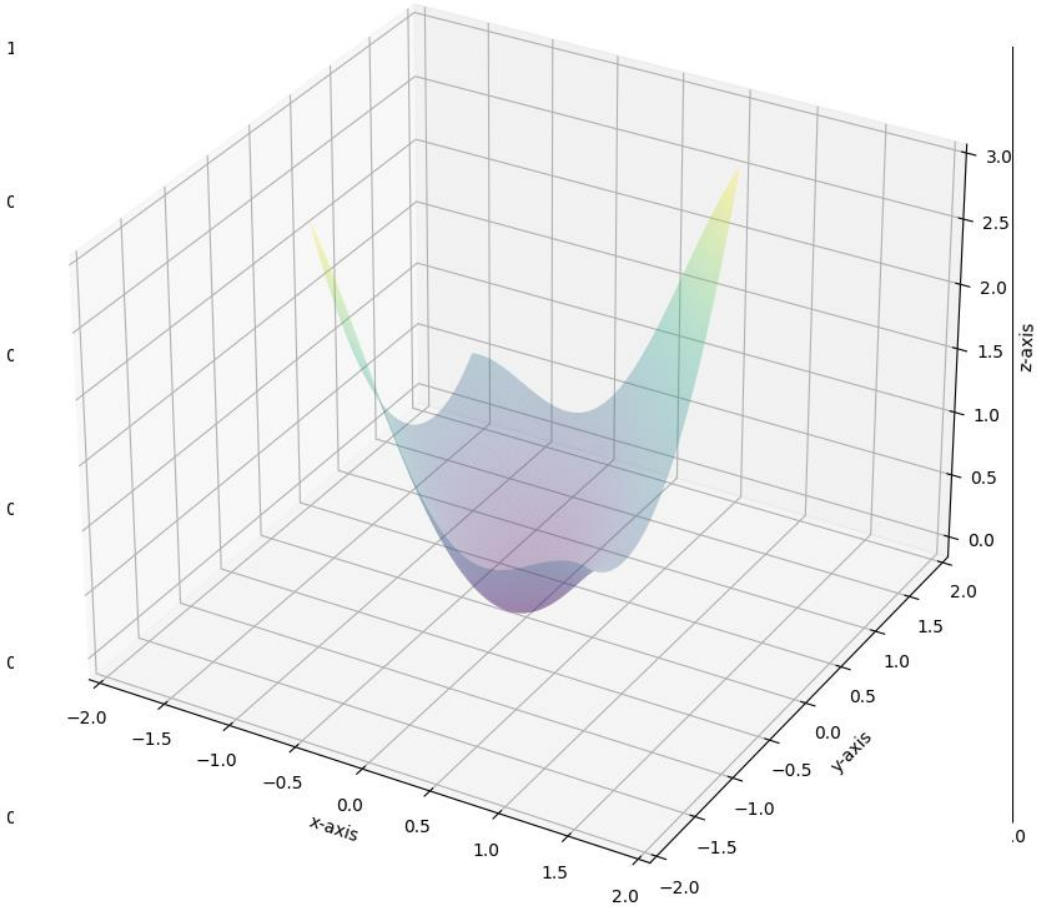

Figure 24. Topology of the Three hump camel function.

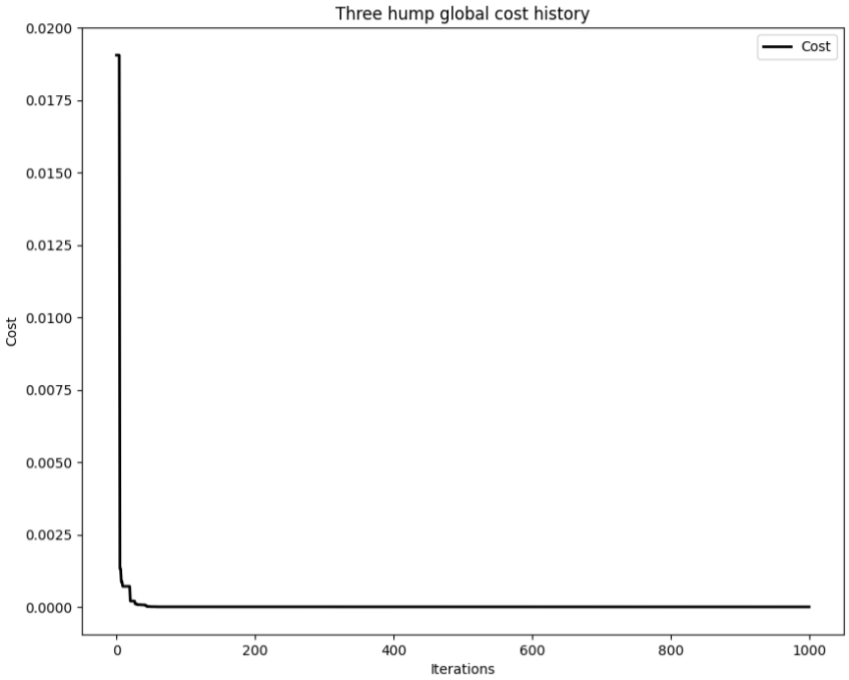

Figure 25. Global best cost history of the Three hump camel function.

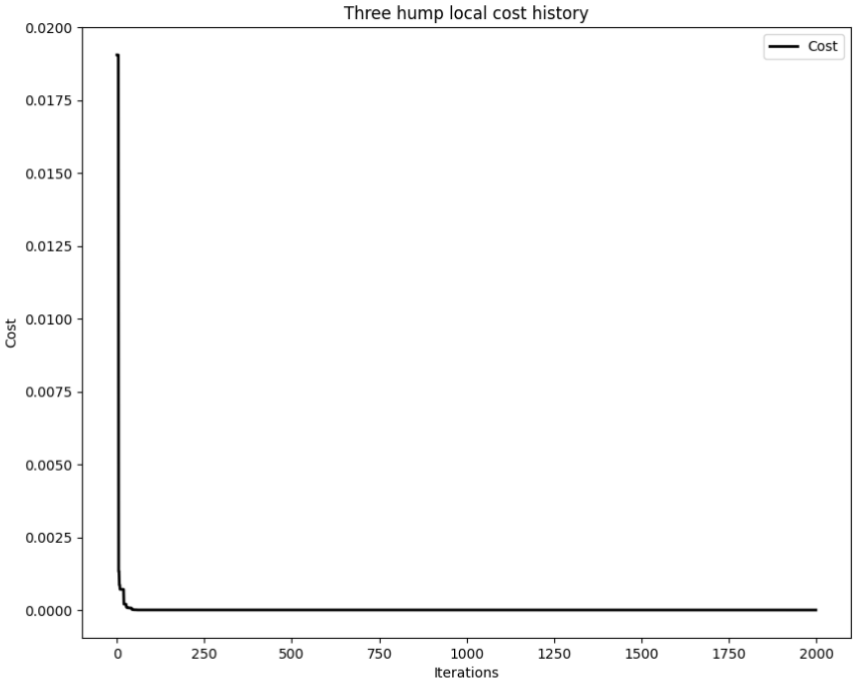

Figure 26. Local best cost history of the Three hump camel function.

Figure 25 shows that the Three hump function produced an optimization in approximately 40 iterations. The results of the optimizations both local and global can be seen in Table 9.

Table 16. Three hump camel global fitness function results.

Global best cost	3.573884008253608e-83
Global best position	[3.91291769e-42, 1.03431390e-42, 2.91372008]

Table 17. Three hump camel local fitness function results.

Local best cost	3.854168552496155e-05
Local best position	[0.00428496, 0.00038943, 0.00398656]
Local best position options	{'c1': 1.5674691186475427, 'c2': 1.6810733419712054, 'w': 1.5794866657386786, 'k': 1, 'p': 1}

Tables 16 and 17 show the corresponding results for global and local optimization. As we can see the global optimization produced a much better result with the cost being magnitudes lower than the local optimization. The global optimization was also derived in fewer iterations than its corresponding local optimization counterpart.

4.11. Summary of Results

When we analyze the results shown in Table 18, we can see that with the exception of Schaffer2 the global best solution consistently produces a better cost value than the local optimization. That is not to say that local optimization does not have a place in a well-rounded application of PSO but instead indicates that the optimum result will be acquired using global optimization instead of local optimization methods. The results showed a visual range of difference in the time taken to execute the various fitness functions. This shows that for example the execution time of the Sphere function is much faster than Levi. Further research can be directed into increasing the performance of some of the slower functions and using machine learning to

determine what cases will benefit from the use of substitute functions when the time taken to execute a given function may negate the benefits its optimization may provide.

Table 18. Global PSO compared to Local PSO.

Fitness function	Best cost	Best position
Global best Ackley	4.440892098500626e-16	[1.45519618e-17, 4.21114928e-16, -1.81346765e-16]
Local best Ackley	0.05203503379902985	[0.00875255, 0.01662873, 0.00552959]
Global best Levi	1.4997597826618576e-32	[1, 1, 0]
Local best Levi	2.9034752985917007e-05	[0.99997622, 0.97846058, 0]
Global best Rastrigin	0.00026	[0, 0, 0]
Local best Rastrigin	0.9180498767704037	[0.02692257, 0.01383459, 0.06131146]
Global best Rosenbrock	4.3651867427948036e-14	[1.00000012, 1.00000021, 0]
Local best Rosenbrock	0.04275490910466276	[0.91951961 0.83882813 0.69599832]
Global best Schaffer2	0.0	[3.90009005e-09 -3.20968894e-07, 3.54832001e]
Local best Schaffer2	4.7471594005754625e-07	[0.01492195, 0.0158504, 0.01532846]
Global best Sphere	0.0	[-1.27259802e-162, -2.48470447e-163, -2.93072783e-163]
Local best Sphere	0.005153627760837311	[0.02606911, 0.06472562, 0.01687078]
Global best Styblinski	-1272.781987190529	[-3.80601064, -0.08326784, -129.68240329]
Local best Styblinski	-1156.6715579144793	[2.15530099, 2.13629179, 122.60836240]
Global best Three hump	3.854168552496155e-05	[3.91291769e-42, 1.03431390e-42, 2.91372008]
Local best Three hump	3.854168552496155e-05	[0.00428496, 0.00038943, 0.00398656]

5. CONCLUSION

In conclusion, swarm optimization can benefit from utilizing different topologies of the swarm to minimize communication costs between particles in the network. Of course, the swarms intended actions must be able to override optimization behaviors when those behaviors contradict with the overall directive of the swarm. Thus, it is important that any automated optimization protocol be able to take into account the swarm's directives and use that information to find the best optimization function to produce a good optimization result. Further research and development in this field will benefit from the development of a system that is designed to recognize and categorize swarm topologies to better enable identification of the best fitness function to employ at any given time as there is no single best solution matching the most optimal fitness function to any given topology that will result in the most efficient optimization. There may also be added benefit from development of transitional fitness functions that bridge the gaps as one topology changes into another. The more quickly optimizations can be derived the faster they can be utilized, and the benefits can become available to the swarm. Global optimizations consistently produced better results overall when compared to local PSO resulting in the best costs. This does not mean that in every case global optimization should be used exclusively but that it would be beneficial to utilize global optimizations to guide the topology and that local optimizations can later be implemented to place particles that may not be able to reach the global optimum. Further research is needed to improve the time taken for some of the slower functions and to categorize them based on the time they will take to execute and produce an effective result to enable a system to utilize them in an effective manner. Another area of future research is the identification of swarm topologies and the predictive systems needed to best optimize organically

changing topologies for example a swarm in flight avoiding obstacles while also maintaining a set formation in real time.

REFERENCES

[1] J. Kennedy and R.C. Eberhart, Particle Swarm Optimization, Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942–1948 ,1995.

[2] M. Guitton. Fighting the Locusts: Implementing Military Countermeasures Against Drones and Drone Swarms. Scandinavian Journal of Military Studies 4.1: 26, 2021.

[3] A. Engelbrecht. Computational Intelligence — An Introduction 2nd Edition. Wiley, 2007.

[4] M. Waibel, B. Keays, Augugliaro, Federico. Drone shows: Creative potential and best practices. ETHzurich research collection. 2017. https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/125498/eth-50527-01.pdf

[5] M. Veltman. Algebraic techniques. Computer Physics Communications, 3:75–78, September 1972.

[6] R. Vetschera. A general branch-and-bound algorithm for fair division problems. Computers and Operations Research, 2010.

[7] S.H. Chen, J. Wu, and Y.D. Chen. Interval optimization for uncertain structures. Finite Elements in Analysis and Design, 40(11):1379–1398, 2004.

[8] A. Eidehall and L. Petersson. Threat assessment for general road scenes using Monte Carlo sampling. 2006 IEEE Intelligent Transportation Systems Conference, 2006.

[9] J. Machta. Strengths and weaknesses of parallel tempering. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 80(5), 2009.

[10] X. Geng, J. Xu, J. Xiao, and L. Pan. A simple simulated annealing algorithm for the maximum clique problem. Information Sciences, 177(22):5064–5071, 2007.

[11] M. Lin and J. Wawrzynek. Improving FPGA placement with dynamically adaptive stochastic tunneling. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29(12):1858–1869, 2010.

[12] C. Wu. Ant colony multilevel path optimize tactic based on information consistence optimize. 2010 International Conference on Computer Application and System Modeling (ICCASM2010), 1:533–536, 2010.

[13] X. Yang and S. Deb. Cuckoo search via levy flights. Proceedings of the 2009 World Congress on Nature; Biologically Inspired Computing (NaBIC 2009), pages 210 – 214, 2009.

[14] C. Pan, C. Xu, and G. Li. Differential evolutionary strategies for global optimization. Shenzhen Daxue Xuebao (Ligong Ban) / Journal of Shenzhen University Science and Engineering, 25(2):211–215, 2008.

[15] A. Engelbrecht. Computational Intelligence — An Introduction 2nd Edition. Wiley, 2007.

[16] J. Digalakis and K. Margaritis. Performance comparison of memetic algorithms. Applied Mathematics and Computation (New York), 158(1):237–252, 2004.

[17] H. Yu-Chi, D. Pepyne. Simple explanation of the no-free-lunch theorem and its implications. Journal of optimization theory and applications 115.3 :549-570, 2002.

[18] PySwarms read the Docs
https://pyswarms.readthedocs.io/en/latest/api/_pyswarms.optimizers.html