Parameter Sensitivity Analysis of Particle Swarm Optimization: A Case Study on Numerical Optimization Problem

Adriana Vlad
Faculty of Computer Science
Computational Optimization and Artificial Intelligence
Iasi, Romania
adriana.vlad157@gmail.com

Andreea-Daniela Lupu
Faculty of Computer Science
Computational Optimization and Artificial Intelligence
Iasi, Romania
dandreaalupu@yahoo.com

***Abstract*—This paper investigates the application of Particle Swarm Optimization (PSO) for numerical optimization problems, focusing on benchmark functions such as Rastrigin, Griewank, Rosenbrock, and Michalewicz across dimensions of 2, 30, and 100. Both plain PSO and a hybrid PSO variant enhanced with a Genetic Algorithm and Hill Climbing (PSO-GA-HC) are evaluated and compared against Genetic Algorithms (GA) and a hybrid GA-HC approach to uncover their strengths and limitations.**

The study incorporates several advanced strategies within the PSO framework to improve its performance. Binary representation is utilized alongside techniques such as dynamic parameter tuning, random shuffle, and adaptive mutation rates. Specifically, the inertia weight (w), cognitive (c_1), and social (c_2) coefficients are dynamically adjusted over iterations to balance exploration and exploitation effectively. Initial tests on the Rosenbrock function showed that smaller w values facilitated faster convergence initially, but stagnated later, while larger values caused erratic movements. A gradual decay of w over iterations mitigated these issues. Similarly, dynamically adjusting c_1 and c_2—starting with $c_1 > c_2$ and reversing this relationship in later iterations—enhanced the algorithm's ability to converge on global optima.

To further enhance performance, elements of Genetic Algorithms were selectively integrated into the PSO framework. Mutation was introduced dynamically, based on the standard deviation of fitness values, while crossover probabilities (p_{cross}) were adjusted to ensure stability during convergence. Hill climbing was selectively applied to the global best particle and random swarm members for localized refinement without overpowering the PSO's global search capabilities.

The impact of swarm size was also analyzed. For the hybrid PSO-GA approach, larger swarm sizes (e.g., $50 \cdot$ num_dims) improved exploration and solution diversity, whereas excessively large swarms (e.g., $100 \cdot$ num_dims) diminished performance due to mismatched parameter configurations. A balanced configuration of $50 \cdot$ num_dims provided consistent and reliable results across benchmark functions.

The results demonstrate that PSO excels on smooth, well-structured functions like Rosenbrock's, particularly when hybridized with genetic algorithms and hill climbing, achieving faster convergence and superior accuracy. On functions dominated by local minima, such as Rastrigin's and Michalewicz's, the robust global exploration of GA outperformed PSO. These findings highlight the importance of parameter tuning, hybridization strategies, and swarm size configuration in tailoring optimization algorithms to specific problem landscapes, paving the way for further advancements in hybrid metaheuristic methods.

***Index Terms*—pso, particle swarm optimization, hill climbing, hybrid, optimization, comparison, genetic algorithm**

I. Introduction

In this paper, the application of Particle Swarm Optimization (PSO) for numerical optimization problems. Its performance on benchmark functions such as Rastrigin, Griewangk, Rosenbrock and Michalewicz is analyzed across dimensions of 2, 30, and 100.

The study evaluates both the results of plain PSO and a hybrid PSO variant enhanced with a Genetic Algorithm and Hill Climbing (PSO-GA-HC), comparing them against Genetic Algorithms (GA) and a hybrid GA-HC approach. By contrasting the outcomes of PSO-GA-HC with GA-HC, we aim to uncover insights into the strengths and limitations of these approaches in addressing challenging numerical optimization problems.

II. Solution Representation

A. Encoding

For both the genetic algorithm and PSO, each solution, known as a *chromosome.x*, is encoded as a binary string. Each chromosome also includes a *chromosome.fitness* value, which represents how well the solution performs in relation to the optimization objective. This fitness value is derived from the function's calculated value, which is stored separately as `chromosome.f` within the chromosome structure. More specifically, since the goal is to minimize all the functions, the fitness is computed as $1/chromosone.f$. The only exception to this is the 4th function, which is negatively defined and thus requires further modification to obtain positive fitness values. For this we add 200 to $chromosome.f$ before computing the inverse.

The binary encoding of *chromosome.x* allows the search space to be discretized to a specified precision. Specifically, a solution's search space, defined by the interval $[a, b]$, is divided into $N = (b-a) \cdot 10^d$ equal subintervals, where d represents the desired decimal precision. To represent all possible $(b-a) \cdot 10^d$ values, we require n bits, where:

$$n = \lceil \log_2(N) \rceil \tag{1}$$

The total length of the bit string encoding a solution is thus the sum of bit lengths required for each parameter of the function being optimized.

Each chromosome's binary string is divided into segments, with each segment corresponding to a parameter in the optimization problem. In order to evaluate the solution (i.e., calculate its fitness), each binary segment has to be decoded into a real value first using the following transformation:

$$X_{\text{real}} = a + \text{decimal}(X_{\text{bits}}) \cdot \frac{(b-a)}{2^n - 1} \tag{2}$$

where:

- X_{real} is the decoded real-valued parameter.
- X_{bits} is the binary segment for the given parameter / dimension.
- $\text{decimal}(X_{\text{bits}})$ converts the binary sequence to its decimal equivalent.
- $2^n - 1$ is the maximum possible decimal value for n bits, ensuring a mapping over the interval $[a, b]$.

B. Number of bits per dimension

Additionally, to represent each dimension of the solution space with a specific precision, we calculate the number of bits required for encoding based on the range of values for each test function. The number of bits per dimension is determined by the following steps:

1) **Retrieving Function Boundaries**: A function is used to retrieve the minimum and maximum values of the domain, $[\min_x, \max_x]$, for the specified optimization function. This defines the search interval for that dimension.
2) **Calculating Discretization**: Based on the desired precision, the number of discrete values needed to span the interval is determined $[\min_x, \max_x]$. This is done by calculating:

   $$N = (\max_x - \min_x) \times 10^{\text{precision}}$$

   where precision represents the required number of decimal places, in our case 5.
3) **Computing the Number of Bits Needed**: To represent all N discrete values in binary, the number of bits is calculated after this formula n :

   $$n = \lceil \log_2(N) \rceil$$

   which gives the minimum number of bits needed to encode each dimension with the desired precision.

This ensures that each dimension can be encoded in binary with sufficient precision, allowing accurate conversion between binary-encoded solutions and their corresponding real values within the search space. In also ensures the minimum sufficient number of bits is used, allowing the program to run as fast as possible while still conforming to the desired precision.

C. Binary Representation in Particle Swarm Optimization (PSO)

As mentioned, in this implementation of Particle Swarm Optimization (PSO), binary representation is used to encode solutions (chromosomes) for storage and processing. This approach leverages the advantages of binary encoding while still performing the core computations of PSO in real-valued space.

1) Before Applying PSO: Before the PSO process begins, each particle is initialized in a binary format. The position of each particle is encoded using a fixed number of bits per dimension, defined by `num_bits_per_dimension`. Random binary values are assigned to each particle's chromosome. This binary representation is converted into real values using the `binary_to_real` function for fitness computation. This ensures the particles' positions are mapped to the problem's real-valued search space.

2) During PSO Execution: While the particles are stored in binary format, the actual computations within PSO are performed in real-valued space. The binary chromosomes are converted into real-valued positions using `binary_to_real`, enabling operations like velocity updates and position adjustments. The updated real-valued positions are then converted back into binary format using the `real_to_binary` function. This hybrid approach ensures compatibility with binary encoding while maintaining the flexibility of real-valued arithmetic.

III. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a nature-inspired metaheuristic algorithm that mimics the social behavior of swarms, such as flocks of birds or schools of fish, to solve optimization problems. In PSO, a population of particles represents potential solutions in the search space. Each particle adjusts its position and velocity iteratively based on its personal experience and the collective knowledge of the swarm. The algorithm is designed to balance exploration and exploitation to efficiently converge to the global optimum.

A. Key Components and Search Strategy

In PSO, each particle tracks two critical values during its search:

- *Personal Best (pBest)*: The best position a particle has visited, representing its highest fitness value observed so far.
- *Global Best (gBest)*: The best position found by the entire swarm, serving as a shared guide for all particles.

The movement of each particle is governed by its velocity, which is updated based on three components:

1) *Inertia*: The particle retains a portion of its previous velocity to maintain its current direction of movement.
2) *Cognitive Component*: The particle is drawn toward its *pBest* to refine known good solutions.
3) *Social Component*: The particle is attracted to the *gBest* to exploit the collective knowledge of the swarm.

The velocity update equation is given by:

$$v_i^{t+1} = w \cdot v_i^t + c_1 \cdot r_1 \cdot (pBest_i - x_i) + c_2 \cdot r_2 \cdot (gBest - x_i), \quad (3)$$

where w is the inertia weight, c_1 and c_2 are the cognitive and social coefficients, and r_1 and r_2 are random numbers uniformly distributed between 0 and 1. The updated velocity is then used to adjust the particle's position as follows:

$$x_i^{t+1} = x_i^t + v_i^{t+1}. \quad (4)$$

The balance between exploration and exploitation is a critical aspect of PSO. Exploration involves searching unvisited areas of the space to avoid local optima, driven by inertia and stochastic randomness. Exploitation focuses on refining solutions in promising regions, guided by the *pBest* and *gBest* tendencies. By tuning the parameters c_1 and c_2, the algorithm can emphasize either individual exploration or swarm-wide exploitation, adapting to the complexity of the optimization problem.

B. Pseudocode for PSO

```
1. Initialize the controlling parameters
   (N, c1, c2, w, vmin, vmax, maxIter)
2. Initialize the population of N
   particles
3. Repeat:
    a. For each particle:
        i. Calculate the objective (
           fitness value) of the particle
        ii. Update pBest if required
        iii. Update gBest if required
    b. Update the inertia weight w
    c. For each particle:
        i. Update the velocity v
        ii. Update the position x
4. Until the stopping condition is
   satisfied
5. Return gBest as the best estimation of
    the global optimum
```

Listing 1. Particle Swarm Optimization (PSO) Pseudocode

C. Strategies in Particle Swarm Optimization and Hybridization Implementation

1) Dynamic Mutation Rate Adjustment: Used during hybridization. Mutation rate adjustment introduces random variations to avoid premature convergence and explore new areas of the search space. The mutation probability is dynamically adapted based on the incline or lack of improvement in fitness.Incline-based adjustments measure the rate of change in the standard deviation of the particle positions across iterations to assess convergence trends. When the incline is positive, it indicates that particles are spreading out, representing the exploration phase of the algorithm. Conversely, a negative incline signals convergence, where the algorithm focuses on exploiting the promising areas of the search space. This strategy dynamically influences the mutation rate and other parameters, allowing the algorithm to balance exploration and exploitation effectively. When incline or fitness stagnation is detected through the incline, the mutation rate increases, enhancing the diversity of the search space.

The mutation rate sometimes also changes every n-th iteration so as to add randomness. This ensures the algorithm avoids getting trapped in local optima and maintains adaptability to complex fitness landscapes.

```
// Adjust mutation rate based on incline
   or iteration
void adjust_mutation_rate(double& pm,
    double incline, double inc) {
     if (incline <= 0.0)
         pm = std::min(pm + 5 * inc,
             0.001);
     else
         pm = std::max(pm - 5 * inc, 0.0);
     pm = std::max(pm - 0.00004, 0.0);
}
//...

//Below is DMRi
if (iter % 100 == 0) {
     adjust_mutation_rate(pm, incline, inc
         );
     //...
}

//...

//Below is DMRstd
if (std_dev <= 0.00001) {
         adjust_mutation_rate(pm, -1, inc
             * 100);
}
```

Listing 2. Example of mutation rate adjusment based on incline or iteration.

2) Dynamic Parameter Tuning: Dynamic adjustment of parameters such as inertia weight, cognitive coefficient, and social coefficient helps balance exploration and exploitation phases. The inertia weight starts high to encourage wide exploration in early iterations and decreases over time to focus on exploitation. The cognitive coefficient begins with a higher value to emphasize individual particle exploration and gradually reduces its influence, while the social coefficient

increases over time to strengthen the focus on the global best solution. These parameter dynamics are implemented using linear decay by taking into consideration the current iteration.

```
if (iter > 15000) {
    w = std::max(0.15 - (iter - 15000) *
        0.00001, 0.1);
    c1 = std::min(1.0 + (iter - 15000) *
        0.0002, 2.0);
    c2 = std::max(2.0 - (iter - 15000) *
        0.0002, 1.0);
}
```

Listing 3. Example of dynamic parameter tuning for w, c1, c2

3) Dynamic Crossover Probability: Used during hybridization. The crossover probability (`pcross`) is dynamically adjusted throughout the iterations to maintain diversity in the swarm population in a step-wise manner. Initially, a high crossover probability promotes exploration, while later iterations see a gradual reduction to emphasize exploitation. Specific thresholds, such as 5000 or 10000 iterations, trigger these changes to adapt to the algorithm's state, ensuring a balanced search process.

```
if (iter > 10000)
    pcross -= 0.00005;
else if (iter > 5000)
    pcross += 0.00005;
```

Listing 4. "DCP" - `pcross` adjusment at specific iteration thresholds.

4) Hill Climbing Integration: Hill climbing is incorporated into the algorithm to complement the global search capability of PSO with fine-tuned local search. This approach is applied to a subset of particles or the global best particle at regular intervals, refining the solutions in the most promising areas of the search space. Hill Climbing is triggered every 100th iteration on the first 5% particles after random shuffling. Afterwards, Hill Climbing is further used on the global best solution, *gbest*, for further refinement. The number of steps in the case of Hill Climbing varies: as the current iteration increases, Hill Climbing becomes progressively more exhaustive.

In every test case, the number of steps is computed dynamically based on the current iteration:

$$\text{Steps} = 5 + \frac{\text{current\_iter}}{500}$$

```
if (iter % 10 == 0) {
    std::random_shuffle(indices.begin(),
        indices.end());
    ThreadPool pool2(num_threads);
    for (int t = 0; t < num_particles; t
        += 2) {
        int r1 = indices[t];
        int r2 = indices[t + 1];
        pool2.enqueueTask([&, r1, r2]() {
            process_population_chunk(
                particles[r1], particles[
                r2], p_best[r1], p_best[r2
                ], temp_particles[r1],
                temp_particles[r2], pcross
                , pm, function_option,
                num_dims,
                num_bits_per_dimension,
                generators[r1 / 2]);
        });
    }
    pool2.~ThreadPool();
}
```

Listing 5. "Rsh" - `std::random_shuffle` to randomize particle indices for chunk processing.

5) Task Parallelization Using Thread Pools: To enhance computational efficiency, thread pools are employed in every test case for parallel processing of particles. By leveraging multi-threading, the workload is divided across available CPU cores, allowing simultaneous updates of particle velocities, positions, and fitness evaluations. This parallelization is particularly beneficial for large populations / large number of particles and high-dimensional spaces, significantly reducing computation time.

LEGEND

- **DMRi**: Dynamic Mutation Rate based on current iteration. The idea behind DMRi is in Listing III-C1. It was applied the same in every test case.
- **DPT**: Dynamic Parameter Tuning. Its application varies greatly.
- **DCP**: Dynamic Crossover Probability. The idea behing DCP is highlighted in III-C3. It was applied the same in every test case.
- **DMRstd**: Dynamic Mutation Rate based on standard deviation. The idea behind DMRstd is in Listing. It was applied the same in every test case. III-C1
- **f1, f2, f3, f4**: Function 1, 2, 3, and 4
- **s2, s30, s100**: Simple versions with 2, 30, and 100 dimensions
- **h2, h30, h100**: Hybrid versions with 2, 30, and 100 dimensions
- **pm** denotes "mutation probability"
- **pcross** denotes "cross probability"

IV. Experimentation Setup

The initial tests for the experimentation were conducted using the Rosenbrock function. To begin, we evaluated the impact of the inertia weight parameter (w) by testing all values between 0.4 and 0.9, incrementing in steps of 0.05. It was observed that smaller w values led to faster initial convergence; however, after this initial phase, the particles would struggle to continue improving and tended to stagnate. Conversely, higher values of w resulted in slower convergence, and excessively large w values caused particles to move too quickly in unproductive directions, often failing to recover and converge effectively.

To address this, we explored research suggesting a gradual decrease of w across iterations. Various strategies were tested, aiming to reduce w to a range between 0.15 and 0.3 by the final iteration. This approach mitigated the limitations of both low and high initial values. For example, excessively small velocities caused similar stagnation issues to those seen with low initial w values, whereas excessively high velocities mirrored the problems of high initial w values.

For the cognitive (c_1) and social (c_2) coefficients, we initially tested equal values for both parameters. This configuration caused particles to struggle as they attempted to move in two conflicting directions simultaneously. Intuitively, and later confirmed through research, we found that setting c_1 higher than c_2 at the start allowed particles to explore their local best solutions more effectively. This helped the swarm converge toward local minima while still maintaining the influence of c_2. However, a large difference between c_1 and c_2 caused faster initial convergence, leading to issues similar to those seen with w.

We also experimented with the sum $c_1 + c_2$. Larger sums facilitated more significant progress with each step but often led to overly large steps, reducing precision. In our final setup, a sum of 3 to 3.5 with a difference of 1 to 1.5 between c_1 and c_2 worked best. During the Rosenbrock function tests, we observed that once the particles reached a valley, it was beneficial to increase c_2 while decreasing c_1, allowing particles to move collectively toward the best point found so far, ideally the global optimum. This adjustment was made to ensure the sum $c_1 + c_2$ remained constant, while their individual values transitioned over iterations, typically flipping from an initial configuration like $c_1 = 2, c_2 = 1$ to $c_1 = 1, c_2 = 2$.

These parameter adjustments were finalized around iteration 5000. However, additional modifications were introduced after these changes to account for iterations beyond this point, leading to an increase in the total number of iterations. For example, we tested scenarios where c_1+c_2 decreased to enable smaller steps toward the end of the optimization process, but this often caused the algorithm to stagnate prematurely. Similarly, starting with $c_2 > c_1$ resulted in faster convergence initially, but this configuration often led to erratic "jumps" in particle behavior, which ultimately performed worse than starting with $c_1 > c_2$.

A. Incorporating Genetic Algorithm Elements

Given the popularity of hybrid PSO-GA-HC approaches in the literature, we incorporated elements of genetic algorithms into our PSO implementation. From the GA algorithm used in our previous numerical optimization research, we selectively included mutation processes but omitted selection to preserve the PSO base structure. Mutation was introduced with a very small initial probability (p_m), which was later adjusted dynamically based on the standard deviation of particle fitness values. This adjustment was made with finer control to ensure the algorithm's stability.

Additionally, since PSO typically requires around 10 times more generations than GA in our previous numerical optimization research, we applied GA operations such as chromosome "processing" every 10 PSO iterations (resulting in GA adjustments every 100 iterations). Mutation and crossover probabilities (p_m and p_{cross}) were fine-tuned to prevent excessive disruption of the PSO convergence. While a high p_{cross} early in the process hindered convergence, gradually increasing it toward the end improved results. Reducing p_{cross} again after this increase allowed the base PSO process to finalize its convergence.

Hill climbing (HC) was also selectively applied, targeting random particles and the global best solution (g_{best}) for occasional refinement. This hybrid approach introduced minor pushes without overpowering the PSO framework, maintaining its identity as the primary optimization algorithm.

B. Impact of Number of Particles

Another key aspect of our experimentation was the effect of the number of particles in the swarm. For the hybrid PSO-GA approach, a larger number of particles proved to be highly beneficial. This aligns with the results observed in our previous experiments on GA numerical optimization, where a larger population allowed for more extensive exploration of the search space. Randomization combined with a larger swarm size enabled the algorithm to evaluate a greater diversity of solutions quickly, significantly improving the overall exploration capabilities.

For the standard PSO, increasing the number of particles also provided some improvement, albeit to a lesser extent. Specifically, we observed that increasing the particle count from $25 \cdot$ num_dims to $50 \cdot$ num_dims led to approximately a 5% improvement in performance. Additionally, the configuration with $50 \cdot$ num_dims exhibited greater consistency across multiple runs.

To further explore this effect, we tested even larger swarm sizes, such as $100 \cdot$ num_dims and $150 \cdot$ num_dims. However, the results with these larger swarm sizes were less favorable. This may be attributed to the fact that the previously optimized parameters, particularly the inertia weight (w), cognitive coefficient (c_1), and social coefficient (c_2), were tuned for a configuration with $50 \cdot$ num_dims. While it is possible that alternative w, c_1, and c_2 values could yield better results for these larger swarm sizes, we opted to retain the balanced

configuration optimized for $50 \cdot$ num_dims, which was already providing consistent and reliable performance.

In summary, the number of particles significantly influenced the performance of the hybrid PSO-GA approach, while its impact on standard PSO was moderate. A swarm size of $50 \cdot$ num_dims offered an effective balance between computational efficiency and optimization performance, establishing itself as a reliable configuration for the parameter set used in this study.

C. Tuned parameters

Function	Num Particles	Max Iterations	pm	pcross
f1s2	$50 \cdot$ num_dims	5000	-	-
f1s30	$150 \cdot$ num_dims	10000	-	-
f1s100	$150 \cdot$ num_dims	2500	-	-
f1h2	$50 \cdot$ num_dims	20000	0.01	0.3
f1h30	$50 \cdot$ num_dims	20000	0.01	0.3
f1h100	$50 \cdot$ num_dims	20000	0.01	0.3

TABLE I
PARTICLE, ITERATION, PM, AND PCROSS SETTINGS FOR FUNCTION 1 ACROSS DIMENSIONS.

Function	DPT1 Variables
f1s2	$w = 0.9 - \text{iter} \cdot 0.0001$ $c_1 = 2.25 - \text{iter} \cdot 0.0003$ $c_2 = 0.75 + \text{iter} \cdot 0.0003$
f1s30	$w = \max(0.8 - \text{iter} \cdot 0.00015, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.000225, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.000225, 2.0)$
f1s100	$w = \max(0.6 - \text{iter} \cdot 0.0002, 0.1)$ $c_1 = \max(2.5 - \text{iter} \cdot 0.0008, 1.0)$ $c_2 = \min(0.5 + \text{iter} \cdot 0.0008, 2.0)$
f1h2	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f1h30	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f1h100	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$

TABLE II
DPT1 VARIABLES FOR FUNCTION 1 ACROSS DIMENSIONS.

Function	DPT2 Variables
f1s30	$w = \max(0.15 - (\text{iter} - 5000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 5000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 5000) \cdot 0.0002, 1.0)$, if iter > 5000
f1s100	$w = \max(0.15 - (\text{iter} - 5000) \cdot 0.00001, 0.1)$ $c_1 = \min(0.5 + (\text{iter} - 5000) \cdot 0.0005, 2.5)$ $c_2 = \max(2.5 - (\text{iter} - 5000) \cdot 0.0005, 0.5)$, if iter > 5000
f1h2	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$, if iter > 15000
f1h30	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f1h100	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$, if iter > 15000

TABLE III
DPT2 VARIABLES FOR FUNCTION 1 ACROSS DIMENSIONS.

Function	Num Particles	Max Iterations	pm	pcross
f2s2	$25 \cdot$ num_dims	5000	-	-
f2s30	$100 \cdot$ num_dims	2500	-	-
f2s100	$50 \cdot$ num_dims	5000	-	-
f2h2	$25 \cdot$ num_dims	20000	0.0	0.1
f2h30	$150 \cdot$ num_dims	20000	0.001	0.3
f2h100	$50 \cdot$ num_dims	20000	0.01	0.3

TABLE IV
PARTICLE, ITERATION, PM, AND PCROSS SETTINGS FOR FUNCTION 2 ACROSS DIMENSIONS.

Function	DPT1 Variables
f2s2	$w = 0.85 - \text{iter} \cdot 0.0001$ $c_1 = 2.25 - \text{iter} \cdot 0.0003$ $c_2 = 0.75 + \text{iter} \cdot 0.0003$
f2s30	$w = 0.6 - \text{iter} \cdot 0.00015$ $c_1 = 2.25 - \text{iter} \cdot 0.00045$ $c_2 = 0.75 + \text{iter} \cdot 0.00045$
f2s100	$w = 0.9 - \text{iter} \cdot 0.00015$ $c_1 = 2.25 - \text{iter} \cdot 0.0003$ $c_2 = 0.75 + \text{iter} \cdot 0.0003$
f2h2	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f2h30	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f2h100	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$

TABLE V
DPT1 VARIABLES FOR FUNCTION 2 ACROSS DIMENSIONS.

Function	DPT2 Variables
f2s2	-
f2s30	-
f2s100	-
f2h2	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f2h30	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f2h100	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000

TABLE VI
DPT2 VARIABLES FOR FUNCTION 2 ACROSS DIMENSIONS.

Function	Num Particles	Max Iterations	pm	pcross
f3s2	$25 \cdot$ num_dims	5000	-	-
f3s30	$25 \cdot$ num_dims	20000	-	-
f3s100	$25 \cdot$ num_dims	5000	-	-
f3h2	$25 \cdot$ num_dims	20000	0.0	0.1
f3h30	$25 \cdot$ num_dims	20000	0.0	0.1
f3h100	$25 \cdot$ num_dims	20000	0.001	0.1

TABLE VII
PARTICLE, ITERATION, PM, AND PCROSS SETTINGS FOR FUNCTION 3 ACROSS DIMENSIONS.

Function	DPT1 Variables
f3s2	$w = 0.85 - \text{iter} \cdot 0.0001$ $c_1 = 2.25 - \text{iter} \cdot 0.0003$ $c_2 = 0.75 + \text{iter} \cdot 0.0003$
f3s30	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f3s100	$w = 0.85 - \text{iter} \cdot 0.0001$ $c_1 = 2.25 - \text{iter} \cdot 0.0003$ $c_2 = 0.75 + \text{iter} \cdot 0.0003$
f3h2	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f3h30	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f3h100	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$

TABLE VIII
DPT1 VARIABLES FOR FUNCTION 3 ACROSS DIMENSIONS.

Function	DPT2 Variables
f3s2	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f3s30	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f3s100	-
f3h2	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f3h30	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f3h100	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000

TABLE IX
DPT2 VARIABLES FOR FUNCTION 3 ACROSS DIMENSIONS.

Function	DPT1 Variables
f4s2	$w = 0.9 - \text{iter} \cdot 0.00015$ $c_1 = 2.25 - \text{iter} \cdot 0.0003$ $c_2 = 0.75 + \text{iter} \cdot 0.0003$
f4s30	$w = \max(0.6 - \text{iter} \cdot 0.0002, 0.1)$ $c_1 = \max(2.5 - \text{iter} \cdot 0.0008, 1.0)$ $c_2 = \min(0.5 + \text{iter} \cdot 0.0008, 2.0)$
f4s100	$w = \max(0.6 - \text{iter} \cdot 0.0002, 0.1)$ $c_1 = \max(2.5 - \text{iter} \cdot 0.0008, 1.0)$ $c_2 = \min(0.5 + \text{iter} \cdot 0.0008, 2.0)$
f4h2	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f4h30	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$
f4h100	$w = \max(0.8 - \text{iter} \cdot 0.0001, 0.15)$ $c_1 = \max(2.0 - \text{iter} \cdot 0.0001125, 1.0)$ $c_2 = \min(1.0 + \text{iter} \cdot 0.0001125, 2.0)$

TABLE XI
DPT1 VARIABLES FOR FUNCTION 4 ACROSS DIMENSIONS.

Function	Num Particles	Max Iterations	pm	pcross
f4s2	50 · num_dims	5000	-	-
f4s30	150 · num_dims	2500	-	-
f4s100	150 · num_dims	2500	-	-
f4h2	50 · num_dims	20000	0.01	0.3
f4h30	50 · num_dims	20000	0.01	0.3
f4h100	25 · num_dims	20000	0.001	0.1

TABLE X
PARTICLE, ITERATION, PM, AND PCROSS SETTINGS FOR FUNCTION 4 ACROSS DIMENSIONS.

Function	DPT2 Variables
f4s2	-
f4s30	$w = \max(0.15 - (\text{iter} - 5000) \cdot 0.00001, 0.1)$ $c_1 = \min(0.5 + (\text{iter} - 5000) \cdot 0.0005, 2.5)$ $c_2 = \max(2.5 - (\text{iter} - 5000) \cdot 0.0005, 0.5)$ if iter > 5000
f4s100	$w = \max(0.15 - (\text{iter} - 5000) \cdot 0.00001, 0.1)$ $c_1 = \min(0.5 + (\text{iter} - 5000) \cdot 0.0005, 2.5)$ $c_2 = \max(2.5 - (\text{iter} - 5000) \cdot 0.0005, 0.5)$ if iter > 5000
f4h2	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f4h30	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000
f4h100	$w = \max(0.15 - (\text{iter} - 15000) \cdot 0.00001, 0.1)$ $c_1 = \min(1.0 + (\text{iter} - 15000) \cdot 0.0002, 2.0)$ $c_2 = \max(2.0 - (\text{iter} - 15000) \cdot 0.0002, 1.0)$ if iter > 15000

TABLE XII
DPT2 VARIABLES FOR FUNCTION 4 ACROSS DIMENSIONS.

V. Test functions

A. Rastrigin's function

Rastrigin's function is defined as:

$$f(\mathbf{x}) = A \cdot n + \sum_{i=1}^{n} \left(x_i^2 - A \cdot \cos(2\pi x_i)\right)$$

where $A = 10$, and n is the number of dimensions.

Solution: The global minimum is at $\mathbf{x} = 0$, where $f(\mathbf{x}) = 0$.

Minima: The function has a large number of local minima, making it challenging for optimization algorithms. The global minimum is at $\mathbf{x} = 0$, where all dimensions are zero.

Results with GA:

TABLE XIII
GENETIC ALGORITHM RESULTS WITHOUT HILL CLIMBING ON RASTRIGIN'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.47070	2.25444	0.00000	0.00000	15.18295	1.64s
30	19.18259	18.05826	0.00029	0.00226	74.05379	44.05s
100	8.76336	10.78267	0.995022	2.81607	95.64831	462.26s

TABLE XIV
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON RASTRIGIN'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.57188	2.44318	0.00000	0.00000	15.21133	2.12s
30	10.52149	12.77225	0.00000	0.00000	46.63114	66.81s
100	102.04804	44.027991	0.00000	1.42432	189.10202	462.26s

Results with PSO:

TABLE XV
PARTICLE SWARM OPTIMIZATION RESULTS WITHOUT GA-HC ON RASTRIGIN'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	0.0	0.0	0.0	0.0	0.887
30	7.7607	2.4960	2.9849	13.9294	215.287
100	83.2780	31.1419	47.7580	167.1528	261.459

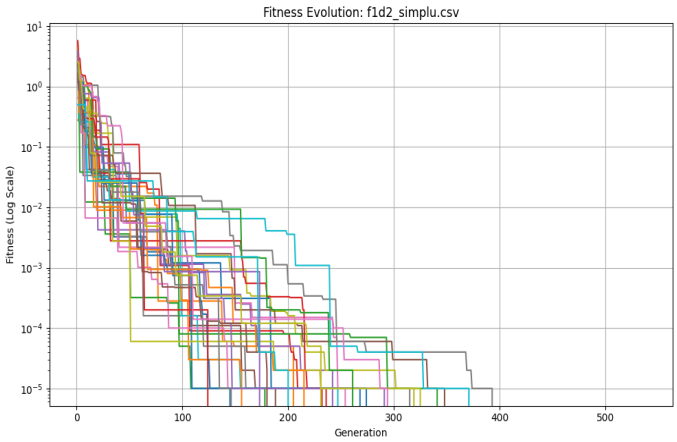

Fig. 1. Particle Swarm Optimization Fitness Evolution Without GA-HC for 2 Dimensions

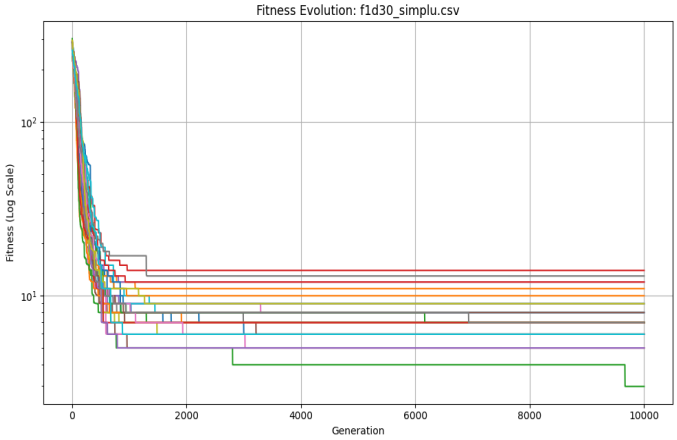

Fig. 2. Particle Swarm Optimization Fitness Evolution Without GA-HC for 30 Dimensions

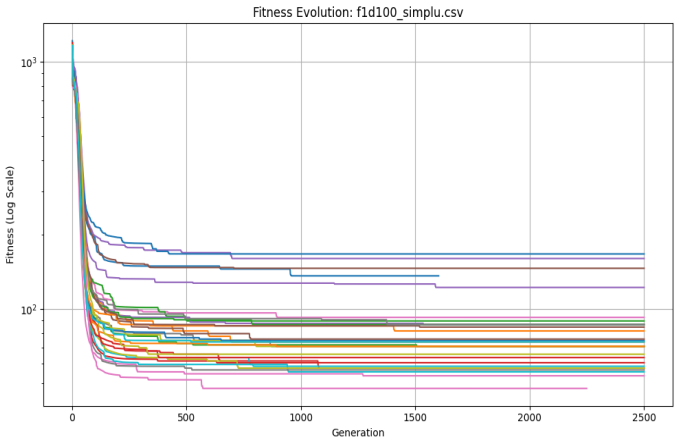

Fig. 3. Particle Swarm Optimization Fitness Evolution Without GA-HC for 100 Dimensions

TABLE XVI
PARTICLE SWARM OPTIMIZATION RESULTS WITH GA-HC ON RASTRIGIN'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	4.5367e-07	3.7620e-07	0.0	9.9e-07	0.409
30	4.5433e-07	3.0996e-07	0.0	9.7e-07	49.284
100	0.1327	0.7144	0.0	3.9798	612.924

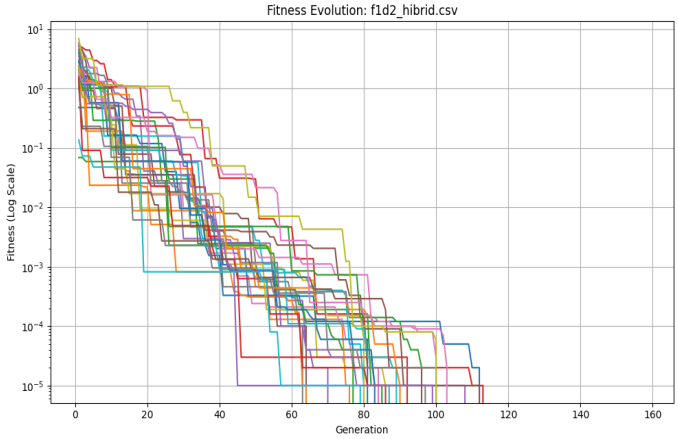

Fig. 4. Particle Swarm Optimization Fitness Evolution With GA-HC for 2 Dimensions

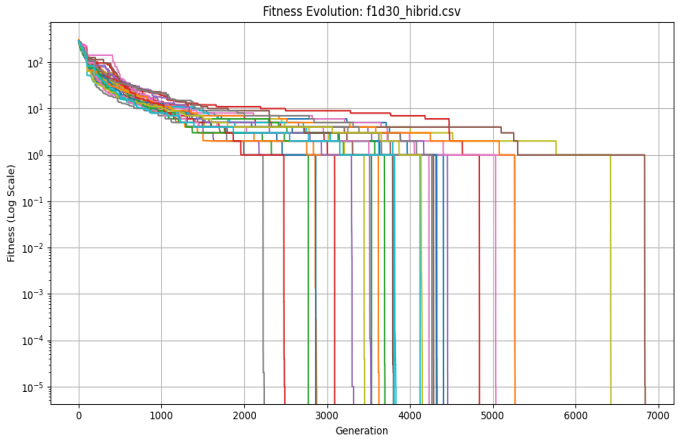

Fig. 5. Particle Swarm Optimization Fitness Evolution With GA-HC for 30 Dimensions

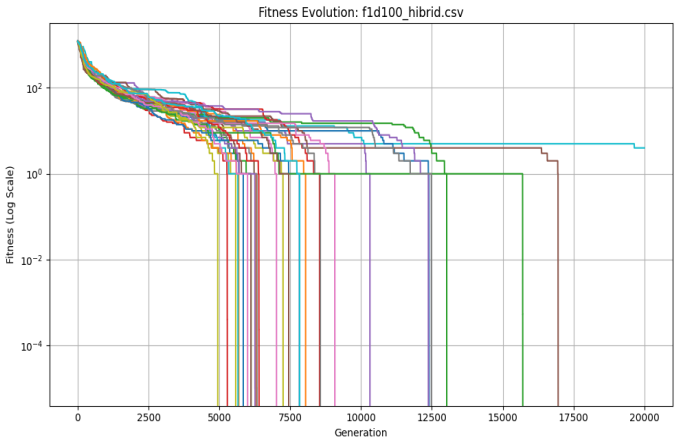

Fig. 6. Particle Swarm Optimization Fitness Evolution With GA-HC for 100 Dimensions

B. Griewank's Function

Griewank's function is defined as:

$$f(\mathbf{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

Solution: The global minimum is at $\mathbf{x} = 0$, where $f(\mathbf{x}) = 0$.

Minima: The function has a relatively smooth landscape with a single global minimum. However, the presence of oscillating cosine terms introduces a complexity that can make it tricky for optimization methods to converge to the global optimum. Results with GA:

TABLE XVII
GENETIC ALGORITHM RESULTS WITHOUT HILL CLIMBING ON GRIEWANGK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.12168	0.43887	0.00000	0.00786	3.23258	1.63s
30	18.22615	31.97513	0.00038	0.00554	130.97456	29.46s
100	3.84867	16.34738	0.00000	0.00026	185.7073	300.64s

TABLE XVIII
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON GRIEWANGK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.14684	0.59445	0.00000	0.00732	4.81149	1.69s
30	15.72083	28.71170	0.00000	0.00041	100.90	57.38s
100	3.13013	11.96393	0.00000	0.00000	165.16862	803.23s

Results with PSO:

TABLE XIX
PARTICLE SWARM OPTIMIZATION RESULTS WITHOUT GA-HC ON GRIEWANK'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	0.00024	0.00132	0.0	0.0074	5.806
30	0.00279	0.00519	0.0	0.0148	85.925
100	0.00104	0.00415	0.0	0.0197	116.921

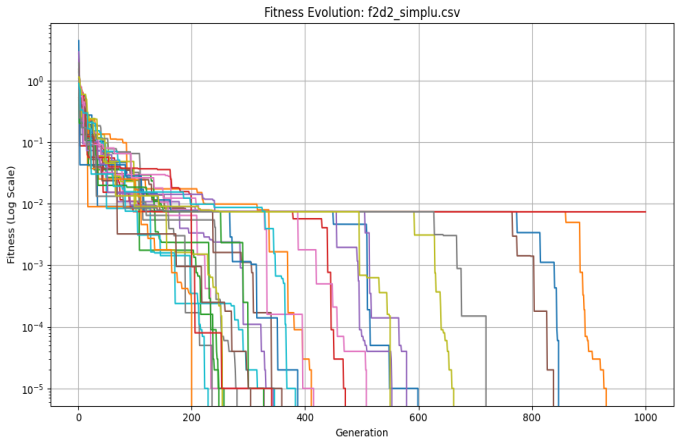

Fig. 7. Particle Swarm Optimization Fitness Evolution Without GA-HC for 2 Dimensions (Function 2)

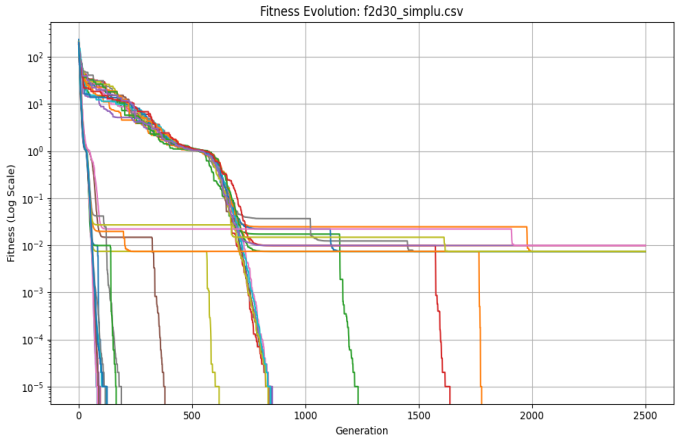

Fig. 8. Particle Swarm Optimization Fitness Evolution Without GA-HC for 30 Dimensions (Function 2)

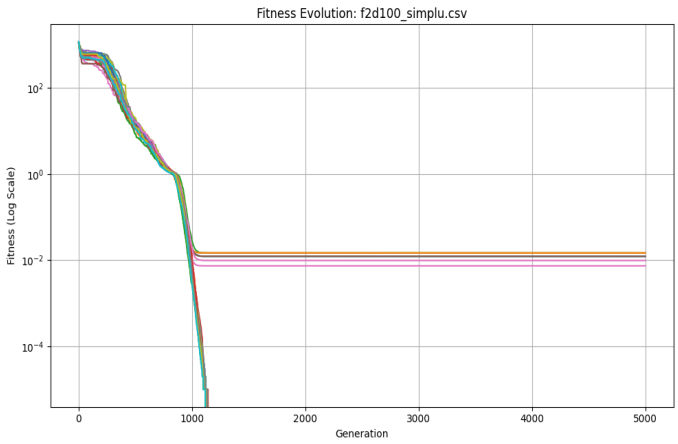

Fig. 9. Particle Swarm Optimization Fitness Evolution Without GA-HC for 100 Dimensions (Function 2)

TABLE XX
PARTICLE SWARM OPTIMIZATION RESULTS WITH GA-HC ON GRIEWANK'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	0.00024	0.00132	0.00000	0.0074	0.868
30	0.00279	0.00519	0.00000	0.0197	85.925
100	0.00000	0.00000	0.00000	1e-06	201.667

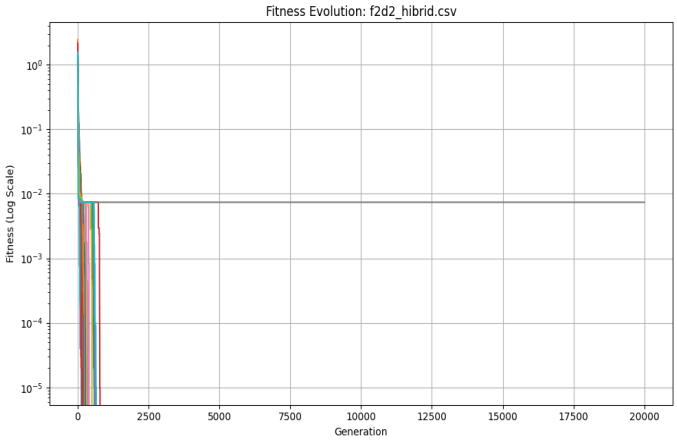

Fig. 10. Particle Swarm Optimization Fitness Evolution With GA-HC for 2 Dimensions (Function 2)

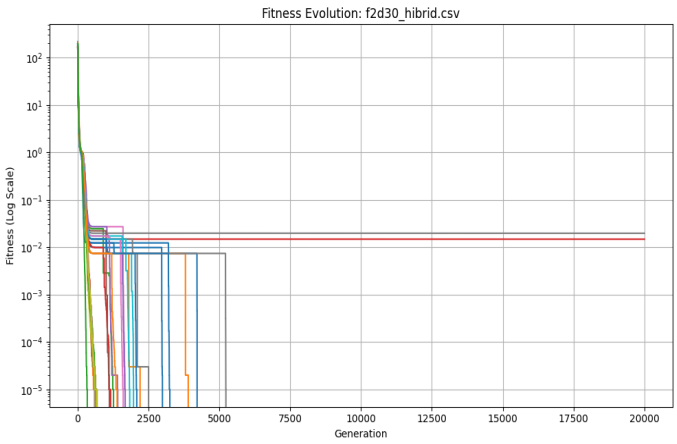

Fig. 11. Particle Swarm Optimization Fitness Evolution With GA-HC for 30 Dimensions (Function 2)

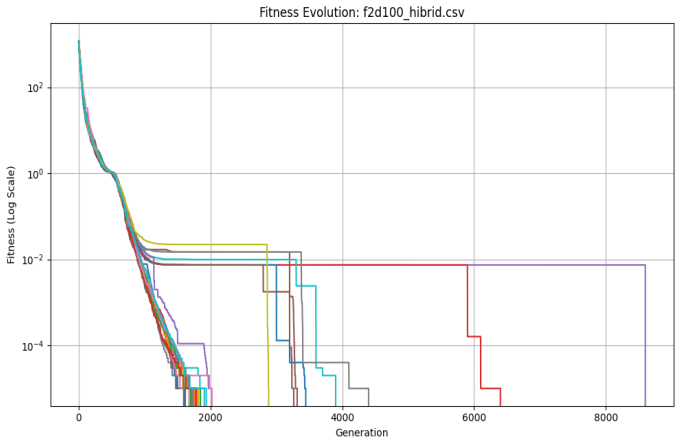

Fig. 12. Particle Swarm Optimization Fitness Evolution With GA-HC for 100 Dimensions (Function 2)

C. Rosenbrock's function

Rosenbrock's function is often called the "Rosenbrock's valley" or "banana function" due to its distinctive shape. It is defined as:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right)$$

Solution: The global minimum is at $\mathbf{x} = (1, 1, \ldots, 1)$, where $f(\mathbf{x}) = 0$.

Minima: The function has a narrow, curved valley containing the global minimum, with many local minima along the valley. Optimization algorithms must carefully navigate this valley to find the global minimum.

Results with GA:

TABLE XXI
GENETIC ALGORITHM RESULTS WITHOUT HILL CLIMBING ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.16722	1.02722	0.00000	0.00032	10.25536	1.61s
30	294.30116	643.08563	23.23250	25.60002	2664.602	23.40s
100	2021.62245	1620.2605	88.7279	95.24692	6403.07893	221.94s

TABLE XXII
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.13664	0.82661	0.00000	0.00014	8.44961	1.66s
30	102.33747	252.01284	21.68510	25.23125	1527.61606	47.37s
100	1720.18480	1492.24736	89.96610	94.41777	5497.15710	916.96s

According to these results, Hill Climbing provides consistency, but not improvement. Therefore for this function also we attempted a different approach: lower population ($100 * dims$), higher initial pm (0.08) an upper limit for pm (0.1), more generations (1500) and more steps for Hill Climbing ($50 + g/50$). While this provided better results for dimension 30, the increase in time complexity made testing for dimension 100 difficult. The average time for a test is about 10 times that of the original approach. What we can say about dimension 100 is that these exact values did not seem to provide the same improvement as they did for dimension 30 in what testing we were able to do. Still, we would like to share these results, as they indicate that a high mutation rate and more thorough Hill Climbing can lead to better final results, albeit in many more generations and much more time. In addition, we have reason to believe that the solution can still be improved after 1500 generations, although the process becomes much slower. As we can see from the graph, there are increasingly long periods of stagnation.

TABLE XXIII
GENETIC ALGORITHM RESULTS WITH IMPROVED HILL CLIMBING ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
30	4882.52953	2170.85052	0.68314	8.280043	9893.247795	424.90s

Results with PSO:

TABLE XXIV
PARTICLE SWARM OPTIMIZATION RESULTS WITHOUT GA-HC ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	3.3333e-10	1.7951e-09	0.0	1e-08	3.86
30	2.9014	2.4568	1.086e-05	7.0982	74.345
100	69.3504	8.2920	47.6923	83.4091	320.699

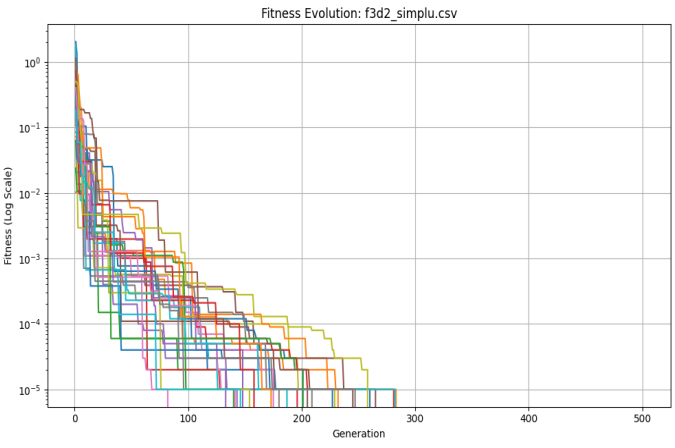

Fig. 13. Particle Swarm Optimization Fitness Evolution Without GA-HC for 2 Dimensions (Rosenbrock's)

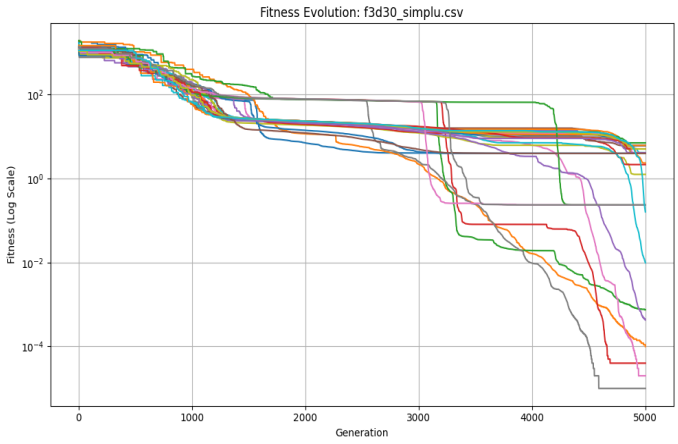

Fig. 14. Particle Swarm Optimization Fitness Evolution Without GA-HC for 30 Dimensions (Rosenbrock's)

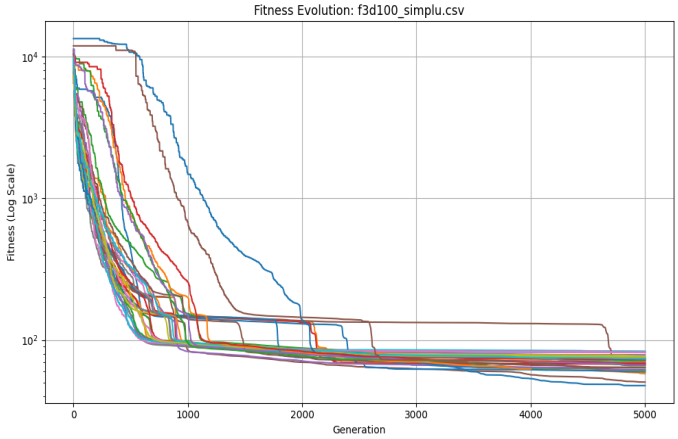

Fig. 15. Particle Swarm Optimization Fitness Evolution Without GA-HC for 100 Dimensions (Rosenbrock's)

TABLE XXV
PARTICLE SWARM OPTIMIZATION RESULTS WITH GA-HC ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	0.00000	2.9782e-07	0.00000	9.9e-07	0.611
30	0.00093	0.00413	0.00000	0.0229	168.106
100	15.4200	12.3890	2.814e-05	38.4408	1118.635

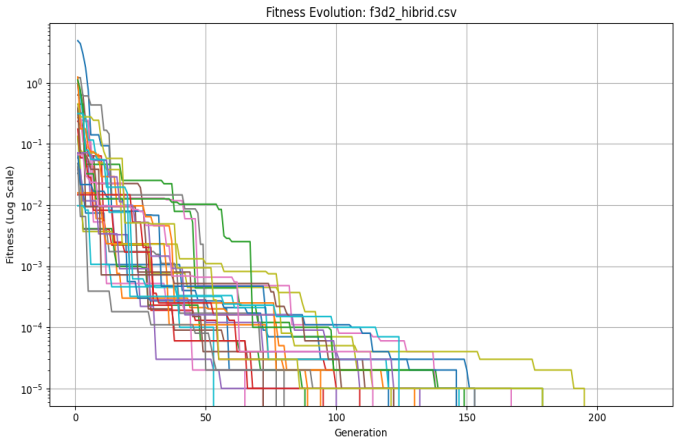

Fig. 16. Particle Swarm Optimization Fitness Evolution With GA-HC for 2 Dimensions (Rosenbrock's)

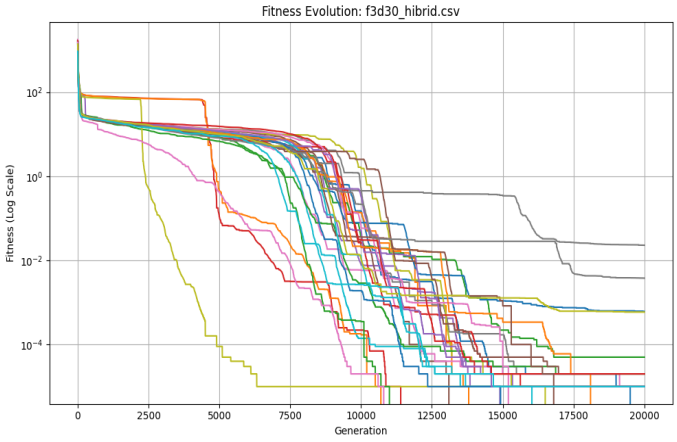

Fig. 17. Particle Swarm Optimization Fitness Evolution With GA-HC for 30 Dimensions (Rosenbrock's)

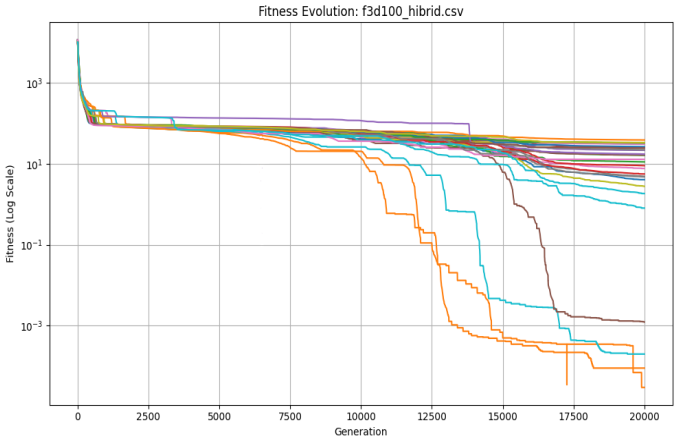

Fig. 18. Particle Swarm Optimization Fitness Evolution With GA-HC for 100 Dimensions (Rosenbrock's)

D. Michalewicz's function

The Michalewicz function is known for its multiple local minima. It is defined as:

$$f(\mathbf{x}) = -\sum_{i=1}^{n} \sin(x_i) \cdot \sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$$

where m is typically set to 10, and x_i is in the range $[0, \pi]$.

Solution: The global minimum depends on the value of n, where n is the dimension. Generally speaking it is slightly bigger than -n. For dimensions 2, 30 and 100 the minimum is -1.8013, -29.6308839 and estimated to be approximately -99.56129, respectively.

Minima: This function has multiple local minima and a single global minimum. Its highly irregular shape with many valleys makes it a challenging problem for optimization algorithms.

Results with GA:

TABLE XXVI
GENETIC ALGORITHM RESULTS WITHOUT HILL CLIMBING ON MICHALEWICZ'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	-1.78875	0.06474	-1.8013	-1.80130	-1.29309	1.63s
30	-28.33641	0.90449	-29.5832	-29.48719	-25.75987	41.51s
100	-97.27816	0.74192	-98.7833	-98.08184	-93.02301	213.67s

TABLE XXVII
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON MICHALEWICZ'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	-1.79007	0.05810	-1.80130	-1.80130	-1.36050	2.15s
30	-29.19724	0.45247	-29.60860	-29.53697	-27.90870	67.84s
100	-91.76821	2.66067	-98.86280	-98.47335	-86.88900	2011.93s

Results with PSO:

TABLE XXVIII
PARTICLE SWARM OPTIMIZATION RESULTS WITHOUT GA-HC ON MICHALEWICZ'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	-1.8013	0.00000	-1.8013	-1.8013	12.979
30	-27.9737	0.6354	-28.8718	-25.8948	53.502
100	-84.3297	4.1461	-89.4437	-69.8670	22.527

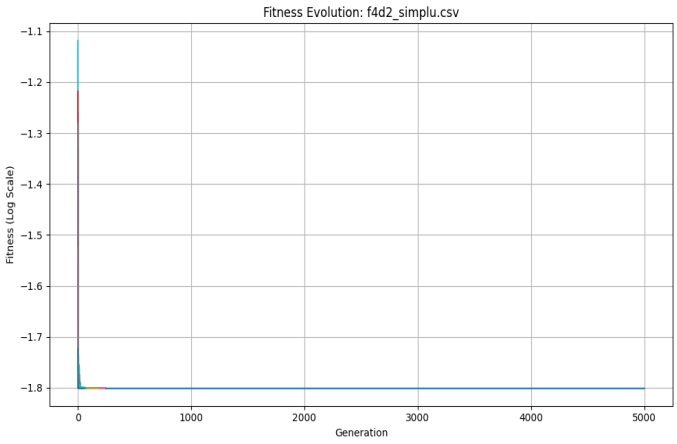

Fig. 19. Particle Swarm Optimization Fitness Evolution Without GA-HC for 2 Dimensions (Michalewicz's)

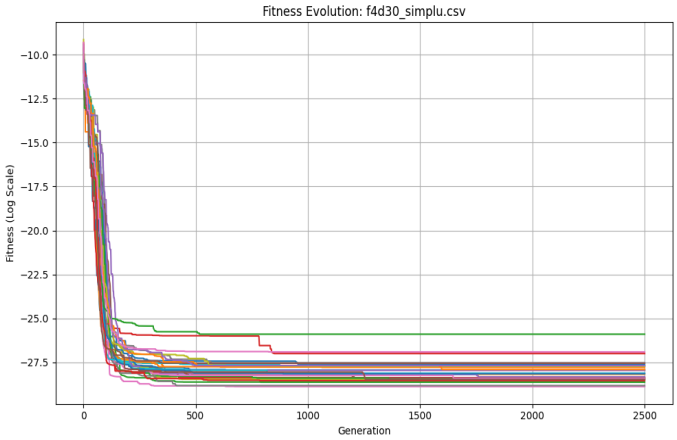

Fig. 20. Particle Swarm Optimization Fitness Evolution Without GA-HC for 30 Dimensions (Michalewicz's)

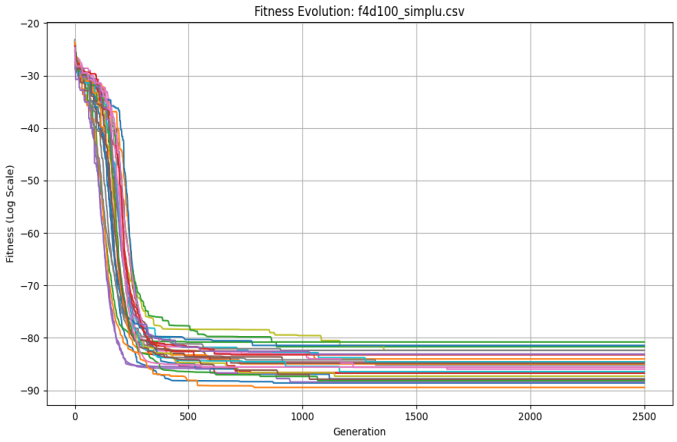

Fig. 21. Particle Swarm Optimization Fitness Evolution Without GA-HC for 100 Dimensions (Michalewicz's)

TABLE XXIX
PARTICLE SWARM OPTIMIZATION RESULTS WITH GA-HC ON MICHALEWICZ'S

Dim	Mean	Std. Dev.	Best	Worst	Time (s)
2	-1.8013	0.00000	-1.8013	-1.8013	1.148
30	-29.4309	0.3474	-29.6309	-28.2361	229.575
100	-93.4214	2.7137	-98.8141	-88.5143	944.377

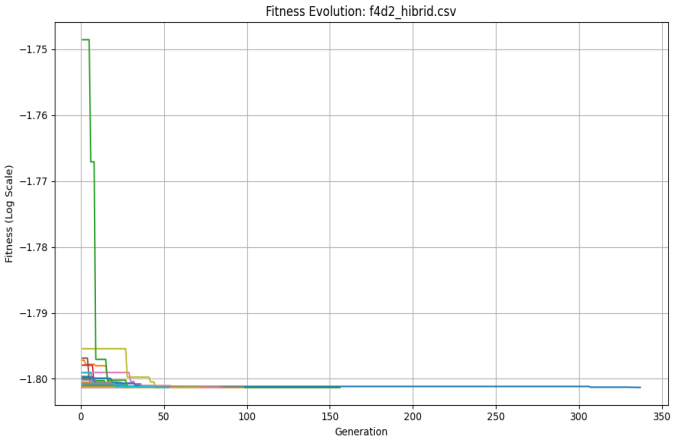

Fig. 22. Particle Swarm Optimization Fitness Evolution With GA-HC for 2 Dimensions (Michalewicz's)

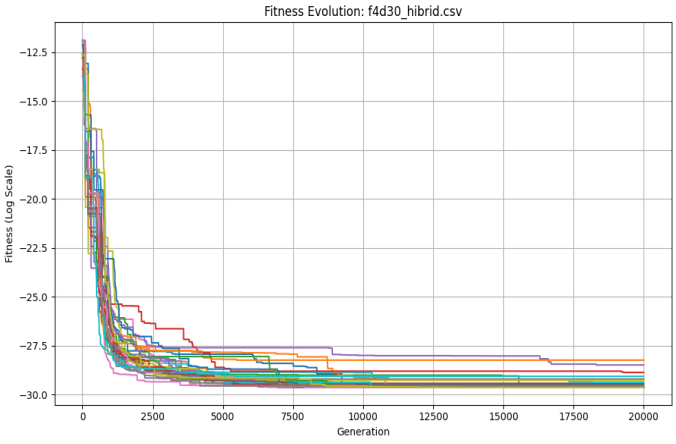

Fig. 23. Particle Swarm Optimization Fitness Evolution With GA-HC for 30 Dimensions (Michalewicz's)

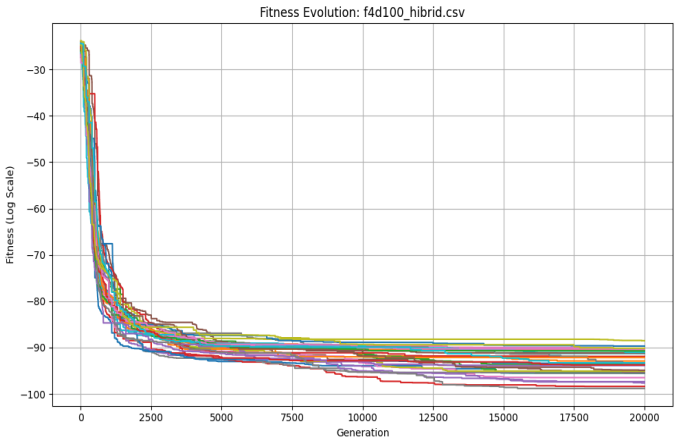

Fig. 24. Particle Swarm Optimization Fitness Evolution With GA-HC for 100 Dimensions (Michalewicz's)

VI. COMPARISON BETWEEN GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION

A. Rastrigin's Function

Rastrigin's function, characterized by multiple local minima, poses significant challenges for both GA and PSO. The results indicate that GA performs more robustly on this function due to its mutation and crossover mechanisms, which help escape local optima. In contrast, PSO struggles significantly with local minima, especially for high-dimensional problems, even in its hybrid version with GA-HC.

- **GA without HC:** For 100 dimensions, GA achieved a mean of 2.81607, a best solution of 0.995022, and a computation time of 462.26 seconds.
- **PSO without GA-HC:** PSO showed a mean of 83.277999, a best solution of 47.75801991, and a computation time of 261.459 seconds.
- **GA with HC:** With hill climbing, GA improved slightly, reaching a mean of 1.42432 and maintaining a best solution of 0.00000.
- **PSO with GA-HC:** Even with hill climbing, PSO exhibited a mean of 0.132661546 and a best solution of 0.0 for 100 dimensions, taking 612.924 seconds.

In summary, GA outperforms PSO on Rastrigin's function due to its ability to avoid premature convergence to local minima.

B. Griewank's Function

Griewank's function, being smoother and less dominated by local minima, proves to be less challenging for both algorithms. However, GA consistently provides superior results in terms of accuracy and convergence speed.

- **GA without HC:** Achieved a mean of 0.00026, a best solution of 0.00000, and a computation time of 300.64 seconds for 100 dimensions.
- **PSO without GA-HC:** Exhibited a mean of 0.00279, a best solution of 0.00000, and a computation time of 85.925 seconds.

- **GA with HC:** Improved further, with a mean of 0.00000 and maintaining a best solution of 0.00000 for 100 dimensions.
- **PSO with GA-HC:** Achieved a mean of 0.00000, a best solution of 0.00000, and a computation time of 201.667 seconds.

Overall, GA's robustness allows it to find better solutions with higher consistency, while PSO benefits from simpler problem landscapes.

C. Rosenbrock's Function

Rosenbrock's function, characterized by a smooth valley leading to the global optimum, is more favorable for PSO. The algorithm benefits from its velocity-driven optimization and exhibits excellent performance when paired with GA-HC.

- **GA without HC:** Results show a mean of 95.24692, a best solution of 88.7279, and a computation time of 221.94 seconds for 100 dimensions.
- **PSO without GA-HC:** Showed a mean of 69.35038, a best solution of 47.69230, and a computation time of 320.699 seconds.
- **GA with HC:** Improved to a mean of 94.41777 and maintained a best solution of 89.96610 for 100 dimensions.
- **PSO with GA-HC:** Outperformed GA, achieving a mean of 15.42000, a best solution of 0.00002, and a computation time of 1118.635 seconds.

For Rosenbrock's function, PSO demonstrates superior performance, particularly when combined with hill climbing.

D. Michalewicz's Function

Michalewicz's function is highly deceptive, with steep local minima that challenge both algorithms. GA performs better due to its global exploration capabilities, while PSO suffers from premature convergence.

- **GA without HC:** Reached a mean of -98.08184, a best solution of -98.7833, and a computation time of 213.67 seconds for 100 dimensions.
- **PSO without GA-HC:** Achieved a mean of -84.32968, a best solution of -89.44372, and a computation time of 22.527 seconds.
- **GA with HC:** Improved results to a mean of -98.47335, a best solution of -98.86280, and a computation time of 2011.93 seconds.
- **PSO with GA-HC:** Achieved a mean of -93.42135, a best solution of -98.81412, and a computation time of 944.377 seconds.

In this case, GA remains more reliable, while PSO's hybrid version shows slight improvements but still struggles with local minima.

VII. Conclusions

The performance of Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) varies significantly based on the characteristics of the benchmark functions. This study compared their performance across four standard benchmark functions—Rastrigin's, Griewank's, Rosenbrock's, and Michalewicz's—while incorporating advanced enhancements like hill climbing (HC) and dynamic parameter tuning for PSO.

Key findings from this study are summarized as follows:

- **Rastrigin's Function**: Due to its numerous local minima, Rastrigin's function poses significant challenges for both algorithms. GA demonstrates superior robustness, leveraging its mutation and crossover mechanisms to escape local optima. PSO, on the other hand, struggles with premature convergence, even when enhanced with GA-HC.
- **Griewank's Function**: A smoother landscape with fewer local minima allows both algorithms to perform relatively well. GA achieves higher precision, whereas PSO remains competitive but exhibits slightly less consistency.
- **Rosenbrock's Function**: This function's smooth optimization landscape favors PSO, particularly with GA-HC, where it achieves rapid convergence and superior results compared to GA.
- **Michalewicz's Function**: With its deceptive local minima, Michalewicz's function presents the greatest challenge. GA outperforms PSO consistently, leveraging its ability to explore globally. PSO, despite enhancements, often gets trapped in suboptimal regions.

Throughout this study, several PSO strategies were implemented to improve its performance on these functions: dynamic parameter tuning, dynamic mutation rate, random shuffle, hybridization with HC etc.

These strategies revealed that while PSO can adapt to different optimization challenges, it remains highly sensitive to the structure of the objective function. Functions with smooth landscapes, such as Rosenbrock's, naturally align with PSO's strengths, while functions dominated by complex local minima require further enhancements.

References

[1] Hussain, A., Ashas, H., Shahid, A., Qureshi, S., Karrila, S. (2024). Hybrid Approach Involving Genetic Algorithm and Hill Climbing to Resolve the Timetable Scheduling for a University. In: Arai, K. (eds) Advances in Information and Communication. FICC 2024. Lecture Notes in Networks and Systems, vol 919. Springer, Cham. https://doi.org/10.1007/978-3-031-53960-2_6

[2] Kennedy, J., Eberhart, R.C. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks IV.

[3] Shi, Y., Eberhart, R.C. (1998). "A Modified Particle Swarm Optimizer". Proceedings of IEEE International Conference on Evolutionary Computation.

[4] Clerc, M., Kennedy, J. (2002). "The Globally Convergent Particle Swarm Optimization Algorithm". IEEE Transactions on Evolutionary Computation.

[5] Eberhart, R.C., Shi, Y. (1996). "Particle Swarm Optimization: Developments, Applications and Resources". Proceedings of the 2001 Congress on Evolutionary Computation.

[6] Zhang, Y., Wang, S. (2010). "An Improved Binary Particle Swarm Optimization Algorithm". Journal of Computational Information Systems.

[7] Mendes, R., Kennedy, J., Neves, J. (2004). "The Fully Informed Particle Swarm: Simpler, Fewer Parameters, Globally Convergent". IEEE Transactions on Evolutionary Computation.

[8] Mitchell, Melanie & Holland, John & Forrest, Stephanie. (1994). When Will a Genetic Algorithm Outperform Hill Climbing?. Advances in neural information processing systems 6.

[9] Wadhwa, A. (2017). Performance analysis of selection schemes in genetic algorithm for solving optimization problem using De jong's function1. Int. J. Recent Innov. Trends Comput. Commun, 5, 563-569.
[10] Chawla, G., & Bala, M. Y. (2014). Solving Optimization Problem By Hybrid Genetic Algorithm Using Hill Climbing In Replacement Operator. Journal of Recent Research Aspects, 2(4), 73-78.
[11] A. Shukla, H. M. Pandey and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), Greater Noida, India, 2015, pp. 515-519, doi: 10.1109/ABLAZE.2015.7154916.