Comparative Analysis of Particle Swarm Optimization and Consensus-Based Optimization in High-Dimensional Spaces

Livia Cardaccia
livia.cardaccia@etu.univ-cotedazur.fr

January 2, 2026

Contents

Chapter 1

Introduction

Optimization problems are fundamental in computer science, engineering, and mathematics, often requiring the minimization of non-convex, high-dimensional objective functions. This report presents a comparative study of two distinct metaheuristic algorithms: **Particle Swarm Optimization (PSO)** and **Consensus-Based Optimization (CBO)**. Both algorithms fall under the category of population-based, stochastic optimization methods, yet they rely on fundamentally different conceptual frameworks.

1.1 Particle Swarm Optimization (PSO)

PSO is a swarm intelligence algorithm inspired by the social behavior of bird flocking or fish schooling. It operates on a population of candidate solutions, referred to as particles, which move through the search space. At a high level, each particle maintains a memory of its personal best position (p_{best}) and is aware of the global best position found by the entire swarm (g_{best}). The movement of a particle is governed by a velocity vector, which is updated iteratively based on three components:

- **Inertia:** The tendency to continue moving in the previous direction.
- **Cognitive Component:** The attraction towards the particle's own historical best position.
- **Social Component:** The attraction towards the swarm's global best position.

PSO relies on the interplay between exploration (searching new areas) and exploitation (refining existing solutions) through these velocity updates.

1.2 Consensus-Based Optimization (CBO)

CBO is a more recent algorithmic class rooted in opinion dynamics and mean-field theory. Unlike PSO, which mimics biological social behavior, CBO draws inspiration from thermodynamics and consensus formation in multi-agent systems. Mathematically, CBO is often described by stochastic differential equations (SDEs). The core mechanism involves computing a weighted average of the particles' positions, the "consensus point", where the weights are determined by the Gibbs-Boltzmann distribution of the objective function values. The update rule consists of two main forces:

- **Drift:** A deterministic force pulling particles towards the weighted consensus point.
- **Diffusion:** A stochastic noise term that scales with the distance from the consensus, allowing for exploration.

While PSO relies on memory (p_{best}), CBO is typically memory-less in its canonical form, relying instead on the instantaneous distribution of the swarm to guide convergence.

Chapter 2

Particle Swarm Optimization (PSO) Implementation

2.1 Algorithmic Workflow

The implementation of the PSO algorithm follows the standard topology where the swarm is fully connected (global best topology). The workflow is iterative:

1. **Initialization:** A population of N particles is initialized with random positions and velocities within the bounds of the search space.
2. **Evaluation:** The objective function is evaluated for each particle.
3. **Update Bests:** Personal bests (p_{best}) are updated if the current cost is lower than the historical cost. The global best (g_{best}) is updated if any particle improves upon the swarm's record.
4. **Velocity Update:** New velocities are calculated using the standard equation:

$$v_{t+1} = w \cdot v_t + c_1 r_1 (p_{best} - x_t) + c_2 r_2 (g_{best} - x_t) \tag{2.1}$$

   where r_1, r_2 are random vectors $\in [0, 1]$.
5. **Position Update:** Positions are updated by adding the velocity vector ($x_{t+1} = x_t + v_{t+1}$). Boundary conditions are handled via clamping.

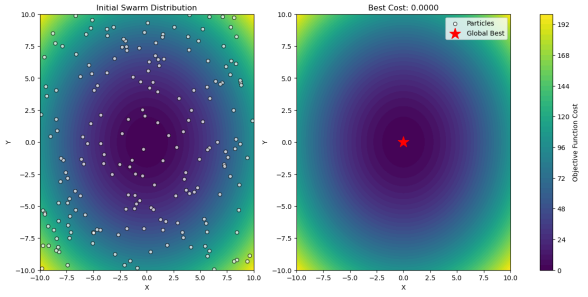
2.2 Parameter Tuning and Numerical Strategy

The performance of PSO is highly sensitive to the inertia weight (w) and acceleration coefficients (c_1, c_2). To achieve convergence in the selected test functions (Sphere, Rosenbrock, Ackley, Schwefel, Rastrigin), an iterative tuning process was employed. Initially, fixed parameters led to stagnation in multimodal functions. The final implementation adopts a **linear decay strategy** for the inertia weight w, decreasing from 0.9 to 0.4 over the course of iterations. This allows for high exploration in the early phases and fine-grained exploitation in the later phases.

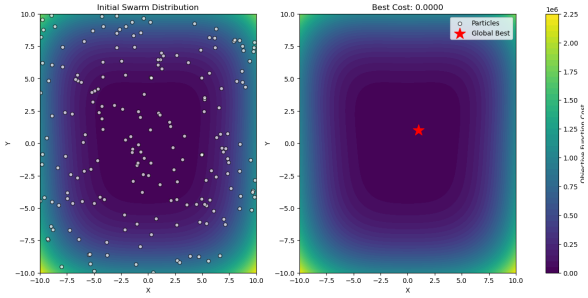
The numerical implementation utilizes vectorization via the NumPy library to ensure computational efficiency, avoiding explicit Python loops over particles where possible.

2.3 Source Code and Visualization

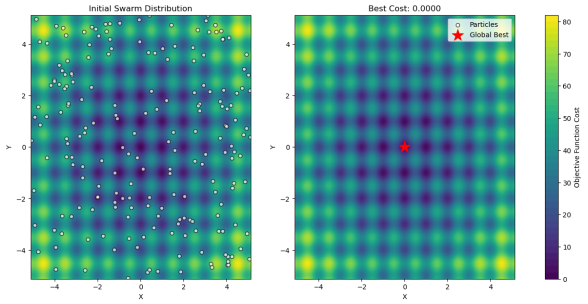
The following figures show the plot of the convergence of the PSO algorithm on the five objective functions, presented as a 2D projection.



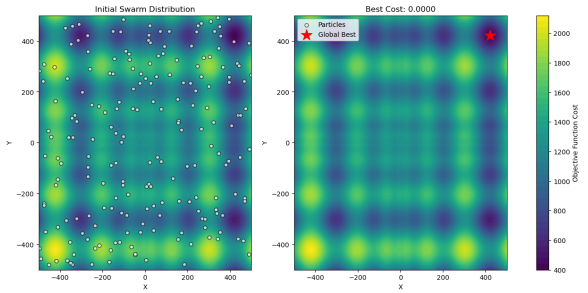
(a) Convergence on the Sphere function.



(b) Convergence on the Rosenbrock function.



(a) Convergence on the multimodal Rastrigin function.



(b) Convergence on the Schwefel function.

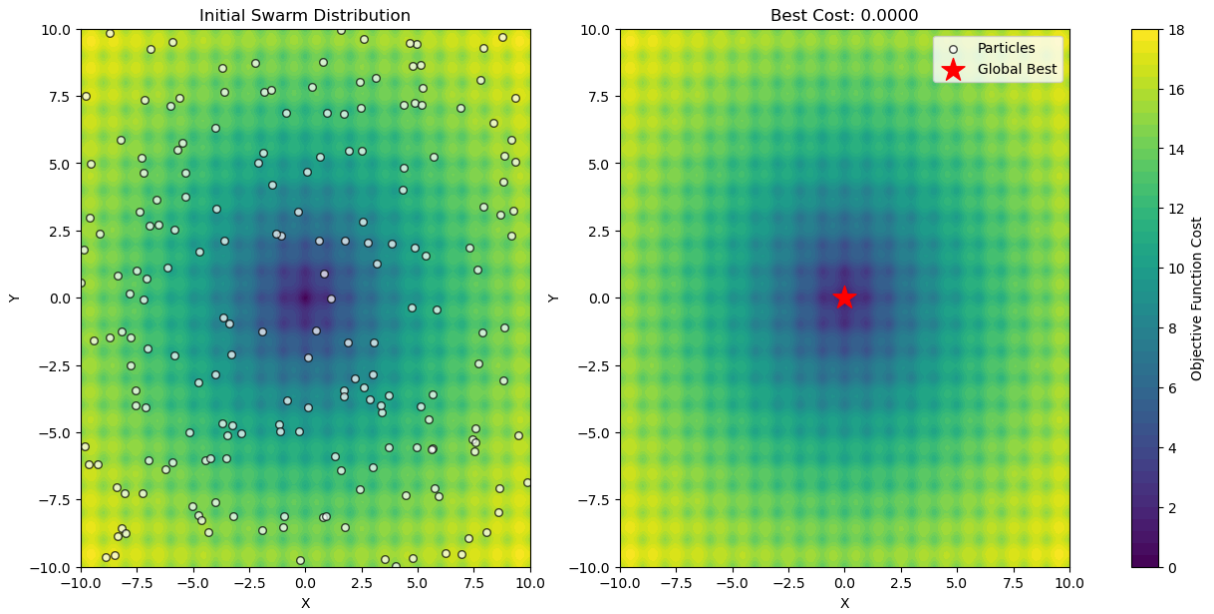


Figure 2.3: Convergence on the Ackley function.

Code Snippet: PSO Velocity Update

```
# Random matrix used to compute the cognitive and social components
r1 = np.random.rand(NUM_PARTICLES, dims)
r2 = np.random.rand(NUM_PARTICLES, dims)

cognitive_component = C1 * r1 * (personal_best_positions - particles)
social_component = C2 * r2 * (global_best_position - particles)

velocities = (w_inertia * velocities) + \
             cognitive_component + \
             social_component
```

Chapter 3

Consensus-Based Optimization (CBO) Implementation

3.1 Algorithmic Workflow

The CBO implementation presented unique challenges compared to PSO due to its sensitivity to numerical scaling. The core workflow involves:

1. **Weight Calculation:** Computing weights $w_i \propto \exp(-\alpha C(x_i))$ where α is a cooling parameter.
2. **Consensus Computation:** Calculating the weighted mean $v_\alpha = \sum w_i x_i$.
3. **Update Step:** Applying the drift term $-\lambda(x_i - v_\alpha)$ and the diffusion term $\sigma|x_i - v_\alpha|dW$.

3.2 Implementation Challenges and Solutions

During the development of the CBO algorithm, several critical issues were identified and resolved to ensure convergence in high-dimensional spaces ($D = 10$).

3.2.1 Numerical Instability (The Shift Trick)

A major issue encountered was **arithmetic underflow**. When calculating weights using $e^{-\alpha C(x)}$, large cost values resulted in effective zeros for all weights, causing division-by-zero errors or loss of gradient information. **Solution:** The "Shift Trick" was implemented, where the minimum cost of the current generation (C_{min}) is subtracted before exponentiation:

$$w_i = \exp(-\alpha(C(x_i) - C_{min})) \tag{3.1}$$

This ensures at least one particle has a weight of $e^0 = 1$.

3.2.2 High-Dimensional Weight Collapse

Even with the shift trick, in 10-dimensional spaces, the magnitude differences in the objective functions (e.g., Rosenbrock or Schwefel) were so large that the exponential function still caused a "weight collapse," effectively assigning a weight of 1.0 to the best particle

and 0.0 to all others instantly. This turned the algorithm into a purely deterministic greedy search. **Solution:** A **Min-Max Normalization** step was introduced. Before computing weights, costs are normalized to the range $[0, 1]$. This keeps the arguments of the exponential function within a reasonable range, preserving the opinion dynamics.

3.2.3 Exploration Freeze vs. Diffusion

Initial implementations utilized a time-decaying diffusion coefficient $\sigma(t) = \sigma \cdot e^{-\gamma t}$. It was observed that this caused the swarm to "freeze" prematurely before locating the global optimum, leading to poor results on multimodal functions. **Solution:** The explicit time decay was removed. In standard CBO, the noise term scales with the distance from the consensus $|x_i - v_\alpha|$. As the swarm converges, this distance naturally approaches zero, reducing noise automatically. This "spatial decay" proved sufficient and more robust.

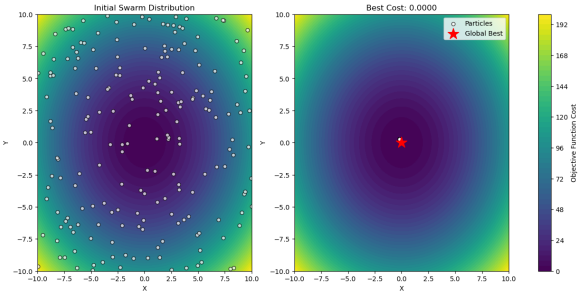
3.2.4 Parameter Tuning History

Significant tuning was required for the balance between Drift (λ) and Diffusion (σ).

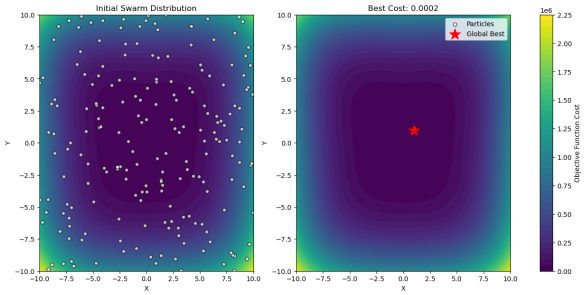
- **Initial Attempt:** $\lambda = 1.0, \sigma = 0.05$. *Outcome:* Failure. The diffusion was too weak compared to the drift, causing instant collapse to local minima.
- **Final Configuration:** $\lambda = 1.0, \sigma = 0.6, N = 200$. *Outcome:* Success. Increasing σ provided sufficient entropic pressure to escape local optima, and increasing the particle count (N) ensured better coverage of the 10D space.

3.3 Source Code and Visualization

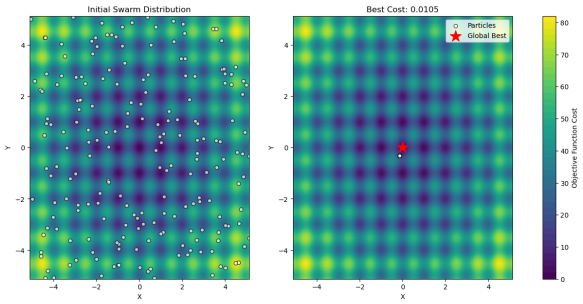
The following figures show the plot of the convergence of the CBO algorithm on the five objective functions, presented as a 2D projection.



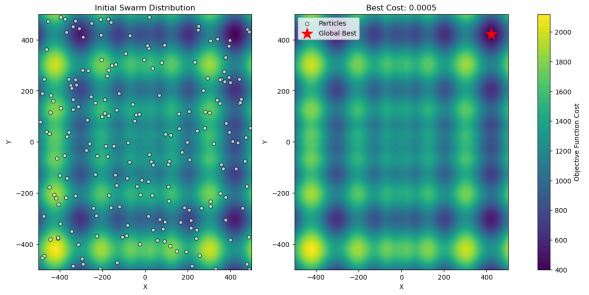
(a) Convergence on the Sphere function.



(b) Convergence on the Rosenbrock function.



(a) Convergence on the multimodal Rastrigin function.



(b) Convergence on the Schwefel function.

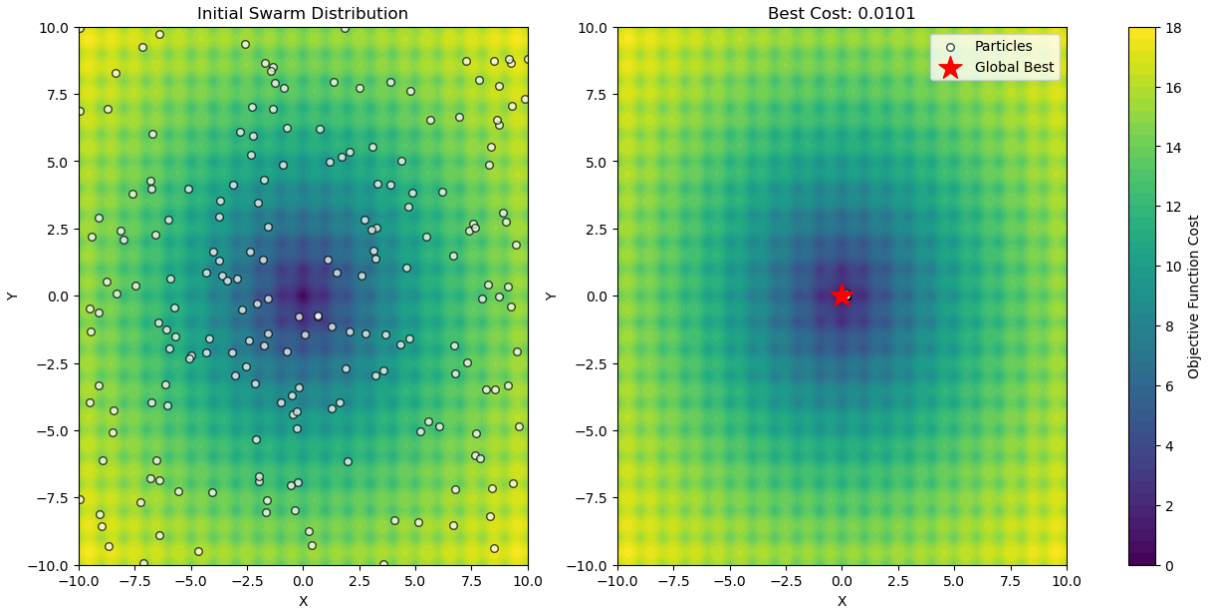


Figure 3.3: Convergence on the Ackley function.

Code Snippet: Normalized Weight Calculation

```
# Update alpha for exploration-exploitation trade-off
progress = iteration / MAX_ITERATIONS
alpha = ALPHA_START + (ALPHA_END - ALPHA_START) * (progress ** 2)

# Update weights based on costs
v_min = np.min(costs)
v_max = np.max(costs)
denom = v_max - v_min if v_max - v_min > 1e-10 else 1.0
norm_costs = (costs - v_min) / denom
weights = np.exp(-alpha * norm_costs)

# Normalize weights
sum_weights = np.sum(weights)
if sum_weights > 0:
    weights /= sum_weights
else:
    # Case all weights are zero (can happen with large alpha)
    weights = np.ones(NUM_PARTICLES) / NUM_PARTICLES

# Calculate consensus point
```

```
consensus_point = np.sum(weights[:, np.newaxis] * particles, axis=0)

# Diffusion vector for each particle
diffusion_vector = particles - consensus_point

# Drift towards consensus point
drift = -LAMBDA * diffusion_vector * DELTA_T

# Distance from consensus point for each particle
dist_scalar = np.linalg.norm(diffusion_vector, axis=1, keepdims=True)

# Standard normal random variables and Diffusion component
zi = np.random.randn(NUM_PARTICLES, dims)
diffusion = SIGMA * dist_scalar * np.sqrt(DELTA_T) * zi

# Update particle positions and keep within bounds
particles += drift + diffusion
particles = np.clip(particles, lower_bound, upper_bound)
```

Chapter 4

Results and Comparison

4.1 Evaluation Methodology

To ensure a fair comparison, the evaluation methodology follows the guidelines proposed in "Best Practices for Comparing Optimization Algorithms" [1]. A **Fixed-Cost** approach was adopted, where both algorithms were given a fixed budget of iterations ($T = 200$ for 10D, $T = 500$ for 2D) and particles ($N = 200$) to minimize the error. The following metrics were collected over 30 independent runs for each function:

- **Success Rate (Reliability):** The percentage of runs that reached the target tolerance (10^{-4}).
- **Final Error (Quality):** The absolute difference between the found solution and the known optimum.
- **Convergence Speed (Efficiency):** Measured by the number of function evaluations required to reach success.

4.2 Quantitative Analysis

The experiments were conducted in both 2-dimensional and 10-dimensional search spaces to evaluate the scalability and robustness of the algorithms.

4.2.1 2-Dimensional Performance

In the low-dimensional case ($D = 2$), the **Particle Swarm Optimization (PSO)** demonstrated excellent robustness, achieving a 100% success rate across all five objective functions (Sphere, Rosenbrock, Ackley, Schwefel, and Rastrigin). The swarm successfully balanced exploration and exploitation to locate the global optima within the iteration budget.

The **Consensus-Based Optimization (CBO)** algorithm also performed well, successfully converging for the Sphere, Ackley, Schwefel, and Rastrigin functions. However, it consistently failed to converge on the **Rosenbrock** function. This specific failure is likely attributed to the Rosenbrock function's topology, characterized by a narrow, curved valley. Unlike PSO, which utilizes an inertia term to maintain momentum along such valleys, the standard CBO's consensus mechanism struggled to navigate the flat valley floor, leading to stagnation before reaching the global minimum.

4.2.2 10-Dimensional Performance

Transitioning to a high-dimensional search space ($D = 10$) revealed significant scalability challenges for both algorithms.

PSO Results: The PSO algorithm maintained its effectiveness on the unimodal Sphere function and the multimodal Ackley function, achieving satisfactory convergence. However, it failed to converge on the more complex Rosenbrock, Schwefel, and Rastrigin functions. In these cases, the swarm typically stagnated in local optima (Rastrigin) or failed to refine the solution sufficiently within the bounds (Schwefel), suggesting that standard parameter sets struggle to escape the vast number of local attractors present in 10D space.

CBO Results: The performance of the CBO algorithm degraded significantly in 10 dimensions. Despite extensive tuning of the drift (λ) and diffusion (σ) parameters to balance the exploration-exploitation trade-off, the algorithm struggled to reach the target tolerance (10^{-4}) for all tested functions. While the algorithm often approached the vicinity of the solution, it lacked the precision to achieve strict convergence. This highlights a critical sensitivity of the implemented CBO variant to the "curse of dimensionality," where the exponential weighting mechanism, even with the Shift Trick and normalization, may inevitably lead to either premature consensus (weight collapse) or excessive diffusion, preventing fine-grained convergence in larger spaces.

4.3 Qualitative Comparison

Based on the experimental data, several qualitative distinctions can be drawn:

- **Robustness in Low Dimensions:** PSO proved to be the more robust algorithm in 2D, handling complex valley structures (Rosenbrock) that baffled the CBO.
- **Scalability:** Both algorithms struggled with scalability, but CBO's performance drop was more drastic. PSO retained utility for specific function types (Sphere/Ackley) in 10D, whereas CBO required highly specific, often unattainable, parameter precision to function.
- **Parameter Sensitivity:** CBO demonstrated extreme sensitivity to parameter tuning in high dimensions. Small changes in σ or α schedules resulted in either complete divergence or stagnation, whereas PSO parameters generally provided a wider stable operating region, despite the limited success on complex 10D landscapes.

Chapter 5

Conclusion

This report presented a comparative implementation of Particle Swarm Optimization and Consensus-Based Optimization. Starting from the theoretical foundations, a robust Python implementation was developed, capable of handling high-dimensional optimization problems.

Key findings include:

1. **Numerical Stability is Key for CBO:** The implementation of the Shift Trick and Min-Max normalization was decisive in making CBO functional in high dimensions. Without these, the thermodynamic weights collapse, making the algorithm useless.

2. **Role of Noise:** In CBO, the balance between drift and diffusion is critical. Unlike PSO, where inertia inherently maintains momentum, CBO relies entirely on the noise term σ to prevent premature consensus.

3. **Suitability:** PSO remains an excellent general-purpose solver for lower-dimensional or "black-box" problems due to its simplicity. CBO offers intriguing properties for problems where a thermodynamic approach to consensus is beneficial, provided the numerical stability is managed.

Future work could investigate Adaptive CBO variants where σ and α are self-tuned based on the population variance, removing the need for extensive manual calibration. The same thing could be done for the PSO algorithm when the dimensionality increases.

Acknowledgments and References

This project was developed with the assistance of several resources:

- The core algorithmic implementations were based on the educational material provided in the YouTube video "Consensus Based Optimization" (`https://youtu.be/FRXsQ6qbJbs`).
- The visualization and plotting structure was inspired by the tutorial "A Gentle Introduction to Particle Swarm Optimization" available at Machine Learning Mastery (`https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optim`
- The Python code for the large-scale massive execution and data collection was generated with the assistance of the AI tool **Gemini** (Google).
- The evaluation metrics and benchmarking methodology were derived from the paper "Best Practices for Comparing Optimization Algorithms" [1].

Bibliography

[1] V. Beiranvand, W. Hare, and Y. Lucet, "Best Practices for Comparing Optimization Algorithms," *arXiv preprint arXiv:1709.08242*, 2017.