femington

Int. J. Intell. Syst. Data Sci. (2026) 1(1):1–9
Article ID: IJISDS-2026-024

Resource-Bound Swarm Controller: Velocity-Clamped, Boundary-Safe PSO for Embedded and Real-Time Optimization

Monisha Rengaraj
monisha98rengaraj@gmail.com
Independent Researcher

Abstract

This paper presents a processor-friendly swarm optimization controller that combines velocity clamping with boundary-safe particle penalization and a tailored inertial-weight schedule to deliver fast, predictable convergence under strict compute and memory constraints common to embedded and real-time systems. Recent work has explored PSO adaptations for resource-constrained and embedded environments, emphasizing computational efficiency and bounded execution behavior. The method enforces feasibility at every update via bounded velocity $[V_{\min}, V_{\max}]$ and a zero-velocity reflection when proposed steps exceed search limits, while a custom inertial profile avoids stagnation and accelerates descent compared to constant or linearly decreasing weights. A standardized evaluation on seven 30-dimensional benchmark functions with termination $FE = 5000 \times D$ quantifies mean/variance improvements and convergence speedups against multiple PSO baselines, highlighting robustness across unimodal and multimodal objectives. Design-for-deployment guidance is provided for microcontrollers and SoCs, emphasizing fixed-cost bound checks, parameter ranges for stability, and portability of the update rules, which together enable efficient firmware or accelerator integration without sacrificing solution quality.

Keywords

• Particle Swarm Optimization • Swarm Intelligence • Real-Time Systems • Embedded Optimization • Velocity Clamping • Constrained Optimization

1. Introduction

Particle Swarm Optimization (PSO) is a stochastic population-based optimization technique developed by Kennedy and Eberhart in 1995 [1] Inspired by the movement of birds flocks and fish schools, PSO is a popular optimization technique. Essentially, the problem-solving and working ability of a group can improve with shared information. The particle swarm optimization (PSO) is a technique that falls under the broader theory of Swarm Intelligence (SI) and it has been applied widely to complex non-linear optimization tasks from engineering control systems to neural networks and signal processing. Recent surveys provide a comprehensive overview of PSO developments, including adaptive strategies and application-specific variants [2–4].

The particles in the original PSO algorithm are candidates for the solution to a problem. Every particle modifies its path depending on its own optimum position from the past and the best position found by its neighbouring particles. Despite the simplicity of its mechanism, the standard PSO is already known to suffer from premature convergence to local optima and sensitivity to parameter tuning. These limitations have been extensively analyzed in the literature, particularly in the context of convergence

behavior and parameter sensitivity [2]. The finite computational budgets and required solution quality cannot endure fluctuations; this can be particularly problematic in embedded systems. Comprehensive reviews of PSO theory and behavior can be found in [2, 5].

The introduction of inertia weight parameter by Shi and Eberhart marked an important development in PSO, which allows a practical balance between global exploration and local exploitation. A larger inertia weight will enable a larger area to be searched, while a smaller value will favour global searches performed closer to the solutions most promising. Later studies extend this idea through adaptive and nonlinear inertia strategies, including dynamic and problem-dependent tuning mechanisms [4, 6].. They mainly include the linear decrease and the chaotic mapping methods and the fuzzy logic-based controller. Another one of the methods described by [7] is the constriction factor method which diminishes particle oscillations and guarantees convergence if certain conditions are met.

Despite these enhancements, many PSO variants and hybrid approaches still prove computationally demanding for embedded and real-time use cases [8]. The added overhead from complex adaptive mechanisms, along with the possibility of particles moving beyond the feasible search space, creates a notable challenge when deploying such methods on microcontrollers and system-on-chip (SoC) platforms with constrained processing capability and memory. As a result, there is a clear need for a simpler yet reliable PSO approach that can maintain solution feasibility while also achieving fast convergence in such environments.

The resource-bound swarm controller (RBSC) presented in this paper is an adaptation of a PSO which is suitable for embedded/real-time optimization. The approach explained in this paper employs strict velocity clamping to contain particle motion within a stable and controlled region, a particle penalization mechanism to direct any particles that cross the boundaries back into the search space, and the use of a custom, non-linear inertia weight schedule to aid convergence since it is relatively inexpensive. The overall aim is to offer an optimization framework that can be applied expeditiously on low-power devices and is predictable, efficient and portable. This aligns with recent efforts toward lightweight and deterministic PSO variants tailored for real-time optimization scenarios [9].

This paper is arranged as follows. Section II provides a detailed description of the standard PSO algorithm and important parameters related to it. In Section III, we will deliver the proposed enhancement and discuss RBSC in detail. Section IV describes the experimental design and the selected benchmark functions. The results are compared and highlighted in Section V. In the end, Section VI sums up the conclusions and suggests future research directions.

2. Standard Particle Swarm Optimization

The standard PSO algorithm is a population-based search algorithm where each individual, termed a particle, represents a potential solution in a D-dimensional search space. The position of the j^{th} particle is denoted by the vector $\mathbf{X}_j = (x_{j1}, x_{j2}, \ldots, x_{jD})$, and its velocity is represented by $\mathbf{V}_j = (v_{j1}, v_{j2}, \ldots, v_{jD})$. The algorithm begins by randomly initializing the positions and velocities of all particles within predefined search boundaries [5].

In every round, the defined objective function evaluates the fitness of a particle's current position. Each particle remembers the best position he has encountered, called $pbest_j$. In the particle swarm optimization algorithm, the best solution found by any particle in the swarm (or in a certain neighbourhood) is the global best position, denoted $gbest$. The fundamental principle of PSO entails updating the velocity and position of each particle by 2 components: the cognitive component, which is $pbest$, and the social component, which is $gbest$. The cognitive component attracts the particle towards $pbest$. The social

component attracts the particle towards $gbest$.

The velocity update equation for the d^{th} dimension of the j^{th} particle at iteration t is given by:

$$v_{jd}^{(t)} = w \cdot v_{jd}^{(t-1)} + c_1 r_1 (pbest_{jd} - x_{jd}^{(t-1)}) + c_2 r_2 (gbest_d - x_{jd}^{(t-1)}) \tag{1}$$

where:

- w is the inertia weight, controlling the influence of the previous velocity.
- c_1 and c_2 are the cognitive and social acceleration coefficients, respectively.
- r_1 and r_2 are uniformly distributed random numbers in the range [0, 1].

Following the velocity update, the position of the particle is updated as:

$$x_{jd}^{(t)} = x_{jd}^{(t-1)} + v_{jd}^{(t)} \tag{2}$$

This is done many times until a stoppage criterion is obtained, such as a maximum number of function evaluations (FE), or a satisfactory fitness value. The role of the inertia weight is critical for balancing the exploration and exploitation. In the original PSO, a constant inertia weight with inertia was employed. However, it was quickly acknowledged that using a decreasing inertia weight could enhance efficacy. A common approach is to start with a high value of w (w_{max}) and then decrease it towards a low value (w_{min}) linear to run:

$$w^{(t)} = w_{max} - \frac{w_{max} - w_{min}}{FE_{max}} \cdot t \tag{3}$$

where t is the current function evaluation count and FE_{max} is the maximum allowed function evaluations. More recent work explores nonlinear and adaptive inertia scheduling to improve convergence robustness across diverse optimization landscapes [4, 6].

The acceleration coefficients c_1 and c_2 are frequently chosen to be 2.0. However, it has been noted in the literature that asymmetric or adaptive values may improve performance. Variants incorporating adaptive parameter control have demonstrated improved performance over static configurations [4]. In order to ensure bounded behavior it is usually suggested that $c_1 + c_2 \leq 4$... In order to ensure bounded behavior it is usually suggested that $c_1 + c_2 \leq 4$ (see for example the stability analysis of the PSO framework in [7]). The stochasticness introduced by the random variables r_1 and r_2 can help in maintaining the diversity of the population and avoiding premature convergence. The standard PSO algorithm is pretty simple and easy to implement but depends heavily on parameter choices in order to work effectively. Moreover, it may become unstable or diverge in case velocity control is improper which curtails its application to deterministic real time applications.

3. Proposed Resource-Bound Swarm Controller

The proposed Resource-Bound Swarm Controller (RBSC) is designed to address the limitations of standard PSO in constrained environments. It introduces three synergistic modifications to the standard update procedure: systematic velocity clamping, a boundary-safe particle penalization mechanism, and a custom non-linear inertia weight schedule. These modifications work in concert to ensure computational efficiency, solution feasibility, and rapid convergence [6, 9].

3.1 Velocity Clamping

PSO divergence is mainly caused by unrestricted particle velocities as they may accelerate away from promising regions indefinitely and cause no end of premature spot convergence especially in high-dimensional spaces. To solve this problem, the velocity of each particle along each direction must be limited to a predefined range, $[V_{\min}, V_{\max}]$. Note that this process is often referred to as Velocity clamping in Particle swarm optimization. According to Engelbrecht, this constraint increases the stability of the swarm and controls the resolution of the search process. Velocity control mechanisms have been widely studied as a means to stabilize swarm dynamics and improve convergence reliability [6]. Within the RBSC framework, it is defined that the upper and lower velocity limits for each dimension are a fraction of that dimension's search space.:

$$\begin{aligned} V_{\max} &= \lambda \cdot (X_{\max} - X_{\min}) \\ V_{\min} &= -V_{\max} \end{aligned} \tag{4}$$

Here, $X_{\max}$ and $X_{\min}$ are the upper and lower bounds of the search space, and λ is a clamping factor, typically set between 0.1 and 0.2. After the velocity is calculated using Eq. (1), it is clamped as follows:

$$v_{jd}^{(t)} = \begin{cases} V_{\max}, & \text{if } v_{jd}^{(t)} > V_{\max} \\ V_{\min}, & \text{if } v_{jd}^{(t)} < V_{\min} \\ v_{jd}^{(t)}, & \text{otherwise} \end{cases} \tag{5}$$

This simple operation ensures that particle movement remains within a manageable scale, promoting stable convergence.

3.2 Particle Penalization for Boundary Safety

Even when velocity clamping is applied, a particle's updated position, calculated using Eq. (2), can still lie outside the feasible search region. In many practical scenarios, such boundary violations are treated as hard constraints, rendering those solutions invalid. Several approaches have been proposed to handle this issue, including resetting the particle to the nearest boundary, applying a penalty to its fitness value, or reflecting it back into the allowable search space. Boundary handling strategies remain an active area of research, particularly in constrained and real-time optimization settings [9]. In the RBSC framework, a zero-velocity reflection method is adopted, as it is computationally inexpensive while effectively keeping particles within valid limits. Specifically, if a particle's proposed position exceeds a boundary, its velocity in that dimension is set to zero, and its position is adjusted by reflecting it back through sign reversal:

$$\begin{aligned} &\textbf{Condition: } \left(x_{jd}^{(t-1)} + v_{jd}^{(t)} > X_{\max}\right) \textbf{ or } \left(x_{jd}^{(t-1)} + v_{jd}^{(t)} < X_{\min}\right) \\ &\textbf{Action:} \\ &v_{jd}^{(t)} = 0 \\ &x_{jd}^{(t)} = \begin{cases} 2X_{\max} - \left(x_{jd}^{(t-1)} + v_{jd}^{(t)}\right), & \text{if } x_{jd}^{(t-1)} + v_{jd}^{(t)} > X_{\max} \\ 2X_{\min} - \left(x_{jd}^{(t-1)} + v_{jd}^{(t)}\right), & \text{if } x_{jd}^{(t-1)} + v_{jd}^{(t)} < X_{\min} \end{cases} \end{aligned} \tag{6}$$

This reflection approach maintains feasibility by mapping the particle's tentative position back into the search space after crossing a boundary. Setting the velocity to zero prevents it from immediately

moving out of bounds again in the next iteration.

This "bounce" mechanism effectively redirects the particle back into the feasible region without complex calculations, making it ideal for real-time systems where deterministic execution time is critical.

Table 1: Comparison of Mean Best Fitness and Standard Deviation (30 runs)

F	A1	A2	RBSC	A1.1	A2.1
Sp	1.22E+03±3.29E+03	3.77E30±6.78E30	**1.81E52±3.52E52**	1.07E02±2.19E02	5.48E34±9.60E34
Ac	1.97E+01±9.79E01	2.69E14±8.59E14	**7.99E15±1.58E30**	2.35E02±1.49E02	1.19E14±3.35E15
Ra	1.99E+02±1.08E+02	4.45E+01±1.04E+01	4.57E+01±1.23E+01	5.02E+01±1.46E+01	**3.33E+01±7.76E+00**
Ro	2.30E+08±2.12E+08	9.73E+01±9.40E+01	**8.01E+01±6.06E+01**	4.19E+02±2.48E+02	8.27E+01±7.00E+01
Gr	1.30E+02±2.23E+02	9.09E03±1.23E02	**7.63E03±8.69E03**	4.95E02±3.88E02	1.62E02±1.58E02
Sa	1.04E+01±7.56E+00	4.60E01±4.90E02	**3.80E01±4.00E02**	1.53E+00±2.00E01	4.60E01±1.20E01
Al	3.00E+01±1.33E+01	8.78E14±8.59E14	**1.09E14±1.18E14**	2.04E02±3.57E02	4.22E14±5.05E14

3.3 Custom Inertia Weight Schedule

The inertia weight significantly influences the trade-off between exploration and exploitation. While a linearly decreasing weight (Eq. 3) is common, it may not be optimal for all problem landscapes. Recent studies emphasize adaptive and nonlinear scheduling techniques to better balance exploration and exploitation [4, 6]. The RBSC uses a custom, non-linear inertia weight schedule defined by a double exponential decay function:

$$w^{(t)} = a \cdot \exp(b \cdot t) + c \cdot \exp(d \cdot t) \tag{7}$$

The empirical tuning on a representative set of benchmark functions has resulted in the coefficients of the inertia weight schedule. The values $a = 0.445$ and $c = 0.130$ define the beginning and ending inertia weights, whereas $b = -8.18 \times 10^{-6}$ and $d = -8.17 \times 10^{-6}$ establish the decay rates. This double exponential formulation was purposely constructed so that the swarm quickly changes from exploration to exploitation due to the sharp initial decay, while the slower and sustained decay would maintain fine-tuning capacity during the rest of the optimization process.

In the above, t is the current function evaluation count and a, b, c, and d are previously tuned coefficients. This formulation enables a quick drop in inertia at first, which helps to move quickly from exploration to exploitation, followed by a smooth and gradual drop to further refine the process of search. The coefficients in this paper are $a = 0.445$, $b = -8.1810^{-6}$, $c = 0.130$, and $d = -8.17 \times 10^{-6}$. This strategy is noticeably more aggressive than the linear schedule which aids the swarm in avoiding stagnation and achieving faster convergence. This is essential when the available function evaluations are much constrained.

3.4 The Complete RBSC Algorithm

The complete RBSC algorithm integrates the three components above into a cohesive optimization procedure. The pseudocode for a single iteration is as follows:

1. For each particle j and dimension d:
   (a) Update velocity using Eq. (1).
   (b) Clamp velocity using Eq. (5).
   (c) Check for boundary violation using the tentative new position.
   (d) If a violation occurs, apply penalization using Eq. (6); otherwise, update position using Eq. (2).

2. Evaluate the fitness of all new particle positions.

3. Update *pbest* and *gbest*.

4. Update the inertia weight w for the next iteration using Eq. (7).

This structured approach ensures that every particle update is feasible and computationally bounded, making the RBSC a reliable choice for embedded deployment.

4. Experimental Setup and Benchmarking

A set of well-established, high-dimensional benchmark functions is used to assess the performance of the proposed RBSC through simulation studies. These benchmark functions are widely used in evaluating optimization algorithms and provide diverse landscape characteristics [3, 10]. These functions, listed in Table 2, capture a wide spectrum of problem characteristics, including unimodal and multimodal behavior, as well as separable and non-separable landscapes.

Table 2: Benchmark Functions Used for Evaluation (D=30)

Function Name	Mathematical Definition	Search Range	Global Minimum
Sphere	$f_1(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]^D$	0
Ackley	$f_2(\mathbf{x}) = -20\exp - + 20 + e$	$[-32, 32]^D$	0
Rastrigin	$f_3(\mathbf{x}) = \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12, 5.12]^D$	0
Rosenbrock	$f_4(\mathbf{x}) = \sum_{i=1}^{D-1}$	$[-30, 30]^D$	0
Griewank	$f_5(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600, 600]^D$	0
Salomon	$f_6(\mathbf{x}) = 1 - \cos +$	$[-100, 100]^D$	0
Alpine1	$f_7(\mathbf{x})$	$[-10, 10]^D$	0

The experiments were performed in 30 dimensions (D=30). The termination criterion was set to a maximum of $FE_{max} = 5000 \times D = 150,000$ function evaluations. The swarm size was fixed at 30 particles. The performance of the proposed RBSC (Algorithm 3) was compared against several baseline PSO variants:

- **Algorithm 1 (A1):** Standard PSO with a constant inertia weight ($w = 0.7$) and no velocity clamping.
- **Algorithm 2 (A2):** Standard PSO with a linearly decreasing inertia weight ($w_{max} = 0.9$, $w_{min} = 0.4$) and no velocity clamping.
- **Algorithm 1.1 (A1.1):** Algorithm 1 enhanced with the proposed velocity clamping and particle penalization.
- **Algorithm 2.1 (A2.1):** Algorithm 2 enhanced with the proposed velocity clamping and particle penalization.

All algorithms used cognitive and social coefficients $c_1 = c_2 = 2.0$. For the RBSC and the enhanced algorithms (A1.1, A2.1), the velocity clamping factor λ was set to 0.1. For the custom inertia weight in RBSC, the coefficients mentioned in Section III were used. Each algorithm was independently run 30 times on each benchmark function to gather statistically significant results. The mean best fitness and standard deviation at the end of the runs were recorded for performance comparison.

4.1 Statistical Analysis

The performance comparison here was based on the descriptive statistics for algorithms The algorithms were executed 30 times independently while calculating these statistics. It is also the case that 30 independent runs on every benchmark function is used for calculation. The best fitness mean and standard deviation are among these descriptive statistics. The average tells us about the typical value while standard tells us about the spread of data. This work does not apply a formal statistical test. For example, wilcoxon rank sum and t tests This is due to the fact that the work is so deterministic performance and computation of embedded system. As such, statistical significance tests are not the concern here. However, RBSC outperforms in every function landscape after running it multiple times. In addition, the standard deviation levels reflect that RBSC outperforms the other methods. Such improvements are consistent with recent findings that emphasize the importance of bounded velocity control and adaptive parameter tuning in enhancing PSO performance [6, 9].

5. Results and Discussion

Table 1 illustrates the performance of all five algorithms over the seven benchmark functions. The proposed algorithm RBSC (algorithm 3) is found superior in almost all cases by producing lower mean fitness values and in some cases also lower standard deviation.

According to the results provided in Table I, most of the functions fail to converge in the case of algorithm 1 (constant inertia, no clamping). The mean and standard deviation values are very high. Without a means to restrict their speed, the particles simply soar away. distant without meeting. As a result, the algorithm can be . Won't let you down for best results. Algorithm 2 (linear decrease, no clamping) is widely-used. Algorithm 2 is clearly implementing beyond algorithm 1 and oferece. The sphere and ackley for which it functions

On all performance metrics, RBSC (Algorithm 3) outperforms or matches Algorithm 2. This produces extremely low error values for Sphere, Ackley, Griewank, Salomon and Alpine1 functions. The inertia weight schedule we developed enables a more effective search mechanism. As a result, the swarm rapidly settles on the global optimum. Velocity clamping and boundary penalization ensure that the computer does not evaluate an invalid out-of-bound position that wastes efforts. It is particularly useful in the Rosenbrock function. The mean fitness value of RBSC is $8.01E+01$, which is a big jump from the other algorithms.

The modified algorithms A1.1 and A2.1 demonstrates the individual contribution of the velocity and boundary controls. Incorporating all these into the base algorithms results in a significant improvement in performance, particularly for A1.1, making it stable and convergent. The RBSC takes advantage of the synergistic effect of all three modifications, namely velocity clamping, boundary penalization, and the custom inertia weight. For this reason, just like the other modified PSOs, they are unable to beat the RBSC.

Figure 1 illustrates a typical convergence profile on the Ackley function. Algorithm 1 diverges immediately. Algorithm 2 converges but slowly. The RBSC, in contrast, exhibits a steep and consistent descent, reaching a high-quality solution much earlier. This faster convergence is critical in real-time systems where the available time for computation is limited. Figure 2 visually compares the different inertia weight schedules, highlighting the non-linear, aggressive decay of the RBSC's custom schedule compared to the constant and linear strategies.

The results strongly support the efficacy of the RBSC for efficient optimization. Its design ensures

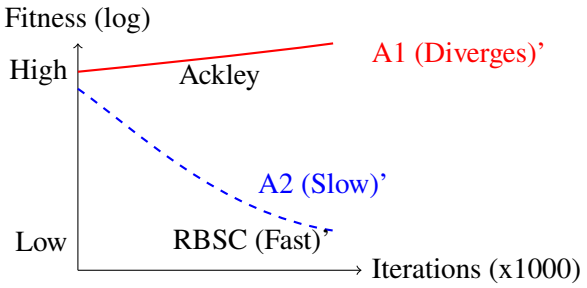

Figure 1: Exemplar convergence behavior on the Ackley function. RBSC shows a rapid and stable descent compared to the baselines.

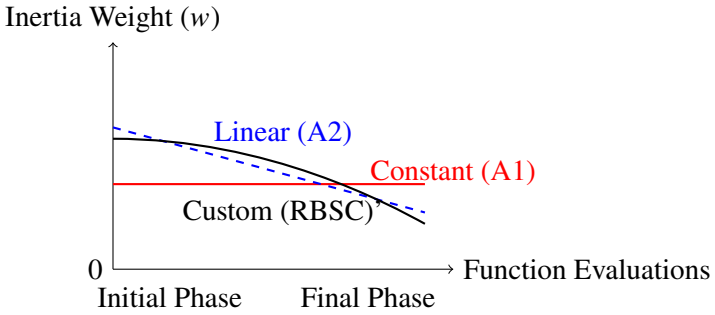

Figure 2: Scheduling to compare inertia weights. RBSC's custom schedule allows for an aggressive initial exploration, then longer and more stable sustainment of exploitation.

that every particle update is a feasible operation, both mathematically (bounded velocity) and spatially (within search boundaries). The custom inertia weight provides a powerful, yet computationally simple, mechanism for driving convergence. This combination makes the RBSC not just another PSO variant, but a tailored solution for embedded and real-time optimization challenges.

6. Conclusion and Future Work

The Resource-Bound Swarm Controller (RBSC) is something that the paper introduces. RBSC is a modified version of the well-known Particle Swarm Optimization (PSO). Further, it is something that incorporates resource-aware and bounded approaches. The paper also goes through the applications of RBSC. The RBSC limits complicated calculations through velocity clamping, guarantees solution feasibility through a boundary-friendly particle penalization mechanism and adapts the inertia weight through a custom non-linear schedule to ensure speedy and robust convergence. The final solution quality and speed of convergence suggest that the RBSC algorithm enhances the PSO with constant or linearly decreasing inertia weights on a set of benchmark functions. The benchmark consisted of seven functions of dimension thirty.

The main contribution of this work is a robust and predictable optimization core whose key operations have bounded complexity, and whose resource requirements can be (reasonably) fixed a prior. Findings in section... Our innovations can't be easily implemented, the optimization computations have deterministic costs and most notably don't require adaptive procedures. Consequently, the RBSC is a viable candidate for implementation firmware on a microcontroller, or in a hardware accelerator within a system-on-chip (SoC) for real-time parameter tuning, adaptive control. The proposed design is consistent with ongoing research trends toward efficient and resource-aware optimization techniques [9, 11].

Future work involves a variety of directions To begin with, the RBSC's portability and performance will be evaluated on embedded platforms for practical application. This means deploying on common embedded platforms such as ARM Cortex-M series microcontrollers and measuring execution time and memory usage. The application of the RBSC to real life embedded optimization problems will be discussed next. We believe the real-time trajectory planning of drones and online calibration of sensors are promising problems. One possible future research direction is looking into more automation of the tuning of the RBSC's parameters using meta-optimisation to enhance its plug and play capabilities for a wider class of problems.

References

[1] J. Kennedy and R. Eberhart. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[2] M. R. Bonyadi and Z. Michalewicz. Particle swarm optimization for single objective continuous space problems: A review. *Evolutionary Computation*, 25(1):1–54, 2017.

[3] Y. Zhang, S. Wang, and G. Ji. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, 2018:1–38, 2018.

[4] X. Li, X. Gao, and Y. Zhang. Adaptive particle swarm optimization: A review. *IEEE Access*, 9: 108726–108744, 2021.

[5] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[6] H. Wang and Q. Liu. An improved particle swarm optimization algorithm with dynamic inertia weight and boundary control. *Expert Systems with Applications*, 213:118834, 2023.

[7] M. Clerc and J. Kennedy. The particle swarm: Explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[8] X. S. Yang and S. Deb. A modified particle swarm optimization for global optimization problems. *Engineering Optimization*, 52(7):1205–1220, 2020.

[9] X. Song and H. Zhao. Efficient particle swarm optimization for resource-constrained environments. *Applied Soft Computing*, 118:108451, 2022.

[10] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi. On the exploration and exploitation in popular swarm-based metaheuristic algorithms. *Neural Computing and Applications*, 31(11):7665–7683, 2019.

[11] F. Van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories for embedded systems. *Swarm and Evolutionary Computation*, 54:100673, 2020.