Article

Soft Actor-Critic Approach to Self-Adaptive Particle Swarm Optimisation

Daniel von Eschwege [1,†] and Andries Engelbrecht [1,2,3,*,†]

[1] Department of Industrial Engineering, Stellenbosch University, Stellenbosch 7600, South Africa; 21785155@sun.ac.za
[2] Computer Science Division, Stellenbosch University, Stellenbosch 7600, South Africa
[3] GUST Engineering & Applied Innovation Research Centre, Gulf University for Science and Technology, Mubarak Al-Abdullah 32093, Kuwait
[*] Correspondence: engel@sun.ac.za
[†] These authors contributed equally to this work.

Abstract: Particle swarm optimisation (PSO) is a swarm intelligence algorithm that finds candidate solutions by iteratively updating the positions of particles in a swarm. The decentralised optimisation methodology of PSO is ideally suited to problems with multiple local minima and deceptive fitness landscapes, where traditional gradient-based algorithms fail. PSO performance depends on the use of a suitable control parameter (CP) configuration, which governs the trade-off between exploration and exploitation in the swarm. CPs that ensure good performance are problem-dependent. Unfortunately, CPs tuning is computationally expensive and inefficient. Self-adaptive particle swarm optimisation (SAPSO) algorithms aim to adaptively adjust CPs during the optimisation process to improve performance, ideally while reducing the number of performance-sensitive parameters. This paper proposes a reinforcement learning (RL) approach to SAPSO by utilising a velocity-clamped soft actor-critic (SAC) that autonomously adapts the PSO CPs. The proposed SAC-SAPSO obtains a 50% to 80% improvement in solution quality compared to various baselines, has either one or zero runtime parameters, is time-invariant, and does not result in divergent particles.

Keywords: particle swarm optimisation; reinforcement learning; soft actor-critic; self-adaptive; swarm intelligence

MSC: 90C26; 90C99; 68W50; 68Q07

check for updates

Citation: von Eschwege, D.; Engelbrecht, A. Soft Actor-Critic Approach to Self-Adaptive Particle Swarm Optimisation. *Mathematics* **2024**, *12*, 3481. https://doi.org/10.3390/math12223481

Academic Editor: Antonio Bolufé-Röhler

Received: 24 September 2024
Revised: 22 October 2024
Accepted: 30 October 2024
Published: 7 November 2024

1. Introduction

Particle swarm optimisation (PSO), introduced by Kennedy and Eberhart [1], is a swarm intelligence algorithm modelled after the swarming behaviour of birds in a flock. PSO finds candidate solutions by iteratively updating the positions and velocities of particles in a swarm. Position and velocity updates for each particle are calculated using the particle's momentum, the best position the particle has visited so far, and the best position visited by any particle in that particle's neighbourhood. PSO is ideally suited to problems with multiple local minima and deceptive fitness landscapes, where traditional gradient-based algorithms fail. PSO performance depends on the use of a suitable control parameter (CP) configuration, which governs the trade-off between exploration and exploitation in the swarm [2–5]. The CP configuration also determines whether the swarm reaches an equilibrium state or diverges. CPs are problem-dependent, and CP configurations that perform well over a large set of problems may not be suitable for the specific problem at hand [3,6], because the exploration-exploitation trade-off is fitness-landscape dependent. However, tuning CPs for a specific problem is computationally expensive.

Self-adaptive particle swarm optimisation (SAPSO) algorithms aim to adaptively adjust CPs during the optimisation process to improve performance, ideally while reducing the number of performance-sensitive CPs [7]. Recent studies have, however, shown that most SAPSO approaches introduce more parameters to which PSO performance is sensitive than they manage to adapt [7,8]. Most SAPSO approaches also result in divergent behaviour, infeasible solutions, low particle step sizes, and are generally ineffective at attaining better solutions [7–9]. Given complex, deceptive optimisation problems, often with rapidly changing landscape characteristics, there is in many cases no basis to assume that what worked up to a certain point will work in the future [10]. If self-adaptation is, therefore, based on behaviour observed up to a certain point during the search, it cannot automatically be inferred that this is indicative of what will work beyond that point. Many attempts also adapt the CP configuration in accordance with the current time-step, making for a time-variant SAPSO algorithm [11]. Time-variant SAPSO is more limited, as it only makes use of the computational budget to inform CP adaptation [12]. Ideally, a time-invariant self-adaptive algorithm should automatically adapt CPs by being fitness-landscape-aware to some extent, and not just depend on the number of time-steps used. Furthermore, particles should neither diverge and leave the feasible search space, nor should they converge prematurely to a local minimum.

Reinforcement learning (RL) is the process through which an agent learns an optimal behavioural policy by trying out actions in an environment, guided by a reward signal [13]. The agent may also receive information in order to know which state the environment is in. An RL agent can, therefore, learn to adjust the CPs of a PSO (action), based on the particle velocities or positions (observation state), guided by the best solution found at the current time-step (reward). The soft actor-critic (SAC) [14] is a specific RL formulation which employs continuous action and state spaces, compatible with the continuous domain of PSO variables, such as particle positions and velocities, and is, therefore, suitable towards dynamic adaptation of PSO CPs during the search process.

The contribution of this study is, therefore, the design and extensive empirical analysis of a truly self-adaptive, RL-based PSO algorithm. The proposed SAC-SAPSO obtains a 50% to 80% improvement in solution quality over relevant performance baselines. The SAC-SAPSO also reduces the number of runtime parameters to either one or zero, is time-invariant, and does not exhibit divergent search behaviour. Finally, whereas RL has been implemented to pre-tune hyperparameters for other heuristics [15–18], no existing research makes use of RL to dynamically adapt CPs during runtime.

The rest of this paper is structured as follows: Section 2 explains the fundamentals of PSO, RL, SAC, and some existing SAPSO algorithms. Section 3 elaborates on design decisions pertaining to the SAC-SAPSO, and Section 4 relates the experimental procedure followed as well as relevant evaluation metrics. Section 5 demonstrates characteristic search behaviour of SAC-SAPSO and discusses the results obtained, Section 6 summarizes the conclusions drawn, and Section 7 outlines future work.

2. Background

This section explains the PSO algorithm, the importance of CPs, methods such as velocity clamping, SAPSO algorithms proposed in the literature, RL principles, and the SAC algorithm.

2.1. Particle Swarm Optimisation

PSO performs swarm-based search to solve optimisation problems by updating particle velocities and then positions on each time-step, based on the best position a specific particle has found (cognitive component), the best position found by the swarm as a whole (social component), and the previous velocity, or inertia, of the particle [1,19]. The velocity update equation per dimension is given as

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_{1ij}(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2ij}(t)\left[\hat{y}_{ij}(t) - x_{ij}(t)\right] \quad (1)$$

where at time t, particle i in dimension j has a velocity of $v_{ij}(t)$, a position of $x_{ij}(t)$, a personal best of $y_{ij}(t)$, and a neighbourhood best of $\hat{y}_j(t)$. The CPs are ω, c_1, and c_2 for the inertia weight, cognitive, and social acceleration components, respectively. Sampled uniformly from (0,1), r_{1j} and r_{2j} introduce stochasticity into the search. Position updates are subsequently calculated per particle i using

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{2}$$

where $\mathbf{v}_i$ is the velocity vector and $\mathbf{x}_i$ is the position vector.

2.2. Control Parameter Configurations

Constant CP values are often selected as $\omega = 0.729844$, $c_1 = 1.496180$ and $c_2 = 1.496180$ [7,20]. However, successful adjustment of the CPs in order to manage the trade-off between exploration and exploitation improves the success of PSO, due to unexplored regions being explored sufficiently, while not wasting computational effort on evidently non-promising areas of the fitness landscape. Usually, more exploration (high ω, high c_1, low c_2) is initially desirable, shifting to exploitative behaviour (low ω, low c_1, high c_2) towards the end of the search. CPs can, therefore, be adjusted in a time-variant fashion using

$$\begin{aligned} \omega(t) &= 0.4\left(\frac{t-n_t}{n_t}\right)^2 + 0.4 \\ c_1(t) &= -3\frac{t}{n_t} + 3.5 \\ c_2(t) &= +3\frac{t}{n_t} + 0.5 \end{aligned} \tag{3}$$

where t is the current time-step and n_t is the maximum number of time-steps (adapted from [21] to satisfy $c_1 + c_2 > 4$ [19]). Similarly to Equation (3), the time-variant acceleration coefficients PSO (PSO-TVAC) [11] also adjusts c_1 and c_2, but keeps ω at a fixed value. Time-variant SAPSO algorithms thus adjust CPs as a function of the current time-step and the total computational budget. Time-invariant SAPSO algorithms, however, adjust CPs as a function of the particle swarm's properties, for example, the average particle velocity, or the rate of improvement in solution quality.

2.3. Convergence Condition

Particles with CPs which adhere to the stability condition [22,23] given by

$$c_1 + c_2 < \frac{24(1-w^2)}{7-5w} \text{ and } w \in [-1,1] \tag{4}$$

are considered stable, since Equation (4) ensures that the particle swarm will in the limit obtain an equilibrium state of zero velocity. This equilibrium state may, however, take very long to achieve, because the stability condition does not make a claim with regards to the number of iterations required to reach this state [24]. Similarly, velocities may also assume very large values during the search process before the equilibrium state is reached.

2.4. Velocity Clamping

Velocity explosion was first observed by Kennedy and Eberhart [19], and is the phenomenon whereby particle velocities grow unboundedly and sometimes leave the search space without ever returning [9]. Velocity explosion is especially common in high-dimensional search spaces, and can be countered using velocity clamping, which imposes limits on the maximum allowable velocity values by introducing the bound v_{max} [25]. Given a search space with boundaries $[\boldsymbol{\ell}, \mathbf{u}]$, velocities can be clamped per dimension by constraining the velocity component in each search dimension j to a fixed value, according to

$$v_{i,j}^{t+1} = \begin{cases} v_{i,j}^{t+1} & \text{if } -v_{\max,j} \leq v_{i,j}^{t+1} \leq v_{\max,j} \\ v_{\max,j} & \text{if } v_{\max,j} < v_{i,j}^{t+1} \\ -v_{\max,j} & \text{if } v_{i,j}^{t+1} < -v_{\max,j} \end{cases} \tag{5}$$

for

$$v_{\max,j} = \delta(u_j - \ell_j),\ \delta \in (0,1) \tag{6}$$

Alternatively, velocity magnitudes can be clamped by setting an upper bound on the magnitude of the velocity vector, i.e.,

$$\mathbf{v}_i^{t+1} = \begin{cases} \mathbf{v}_i^{t+1} & \text{if } \left\|\mathbf{v}_i^{t+1}\right\| <= v_{\max} \\ \frac{v_{\max}}{\left\|\mathbf{v}_i^{t+1}\right\|}\mathbf{v}_i^{t+1} & \text{if } \left\|\mathbf{v}_i^{t+1}\right\| > v_{\max} \end{cases} \tag{7}$$

for

$$\begin{aligned} v_{\max} &= \delta\sqrt{\sum_{j=1}^{n_x}(u_j - \ell_j)^2} \\ &= \delta|\mathbf{u} - \boldsymbol{\ell}| \end{aligned} \tag{8}$$

where $\mathbf{u} = [u_1, \ldots, u_{n_x}]^T$ and $\boldsymbol{\ell} = [\ell_1, \ldots, \ell_{n_x}]^T$ are n_x-dimensional vectors.

Whereas clamping by magnitude purports to having the advantage of maintaining directionality, since all velocity components are scaled by the same factor, it has the disadvantage that large velocity components dominate, and small components might become infinitesimal. Conversely, velocity clamping by dimension modifies the particle's trajectory, but smaller velocity components maintain significance. However, since the random components $\mathbf{r}_1$ and $\mathbf{r}_2$ modify the particle's trajectory in all dimensions anyway, velocity clamping per dimension is preferred here.

2.5. *Self-Adaptive Particle Swarm Optimisation*

Numerous SAPSO algorithms exist in the literature, but many of them lead to particles that diverge into infeasible search space, or premature convergence due to step sizes which quickly approach zero [8]. Additionally, most of these algorithms add more CPs than are originally present. This observation is supported by another study [7], which found that many of the 18 examined SAPSO algorithms either displayed divergence or premature convergence. Efforts to suitably adjust the CPs ω, c_1, and c_2, therefore generally fail to do so in a way that reduces the number of parameters which impact performance. Exceptions are SAPSO by [26], improved PSO by [27], and PSO with random acceleration coefficients by [7].

This section discusses SAPSO algorithms relevant for the purposes of this study.

2.5.1. Adaptive Particle Swarm Optimisation Based on Velocity Information

Adaptive particle swarm optimisation based on velocity information (APSO-VI) [28] seeks to adjust particle velocities towards an optimal, decreasing velocity by manipulating the inertia weight ω to reflect the shift from exploration to exploitation. The APSO-VI algorithm functions as follows: the average velocity of all particles in the swarm is calculated using

$$v_{avg}(t) = \frac{1}{n_x n_s}\sum_{i=1}^{n_s}\sum_{j=1}^{n_x}|v_{ij}(t)| \tag{9}$$

where n_x is the number of dimensions and n_s is the size of the swarm. Thereafter, the optimal velocity is defined as

$$v_{ideal}(t) = v_s\left(\frac{1 + \cos\left(\pi\frac{t}{T_{0.95}}\right)}{2}\right) \tag{10}$$

with the initial optimal velocity v_s as

$$v_s = \frac{x_{max} - x_{min}}{2} \tag{11}$$

and $T_{0.95}$ is the point where 95% of the search is finalized; x_{max} and x_{min} are the respective boundaries of the feasible search space. The CP ω is then updated using

$$\omega(t+1) = \begin{cases} \max\{\omega(t) - \Delta\omega, \omega_{\min}\} & \text{if } v_{avg}(t) \geq v_{\text{ideal}}\,(t+1) \\ \min\{\omega(t) + \Delta\omega, \omega_{\max}\} & \text{if } v_{avg}(t) < v_{\text{ideal}}\,(t+1) \end{cases} \tag{12}$$

where $\omega_{min} = 0.3$ and $\omega_{max} = 0.9$ are the minimum and maximum inertia weights, and $\Delta\omega = 0.1$ is the inertia weight step size. As the search progresses, the decrease in ω causes the CP configuration to become convergent according to Poli's convergence criterion in Equation (4). Consequently, while initially almost the whole swarm resides in infeasible search space, upon completion of the search, only 6% of particle positions are infeasible.

APSO-VI is of interest, because it reflects the intuitive expectation that a successful SAPSO algorithm should exhibit a transition from exploration towards exploitation as the search progresses. Whether this characteristic is automatically learned by an RL-based SAPSO algorithm might be informative.

2.5.2. Adventurous Unified Adaptive Particle Swarm Optimisation

Adventurous unified adaptive particle swarm optimisation (UAPSO-A) [29] makes use of learning automata, which are defined as adaptive decision-making devices. As the learning automata interact with their stochastic environment, they begin to make better decisions, in this case choosing new CP values for the next time-step.

The UAPSO-A algorithm functions as follows: for each CP, a separate learning automaton selects new CP values from a discrete, predefined range of values on each time-step. The inertia weight range consists of n_ω equidistant values in $[\omega_{\min}, \omega_{\max}] = [0, 1]$. The cognitive and social components' range consist of n_c equidistant values in $[c_{\min}, c_{\max}] = [0, 4]$. A feedback signal instructs the automaton. If selection of the CP value at index i improved solution quality, selection probabilities are updated using

$$p_j(t+1) = \begin{cases} p_j(t) + a(1 - p_j(t)) & \text{if } i = j \\ p_j(t)(1 - a) & \text{otherwise} \end{cases} \tag{13}$$

where p_j is the selection probability of the CP value at index j. Furthermore, a is the reward step size, which increases the likelihood of reselection for the successful CP value and decreases the likelihood for all other CP values. If selection of the CP value at index i did not improve the solution quality, selection probabilities are updated using

$$p_j(t+1) = \begin{cases} p_j(t)(1 - b) & \text{if } i = j \\ \frac{b}{|A|-1} + p_j(t)(1 - b) & \text{otherwise} \end{cases} \tag{14}$$

where b is the penalty step size, and $|A|$ is the number of possible CP values to choose from. The likelihood of reselection of the unsuccessful CP value is, therefore, decreased.

UAPSO-A demonstrates convergent behaviour despite not enforcing Poli's convergence criterion in Equation (4). Between 60% and 80% of particles have convergent CP values at any given time-step. As a result, many particles initially violate boundaries of the feasible search space, but the bound violations decrease smoothly to less than 40% towards the end of the search process as the parameters are adapted [7]. Finally, UAPSO-A adapts ω, c_1, and c_2, but introduces nine constants, namely, n_ω, $\omega_{\min}$, $\omega_{\max}$, n_c, $c_{\min}$, $c_{\max}$, a, b, and τ. UAPSO-A is of interest because the learning automata can be considered as a rudimentary form of RL. RL similarity lies in that the automata take actions by making decisions, informed by a reward signal which is based on the objective function value of the particle.

2.6. Reinforcement Learning

RL is the process by which an agent learns an optimal action to take in a given situation, guided by a reward signal [13]. In many cases, the agent observes the environment, meaning that it receives information that conveys the current state of the environment. The agent learns by interacting with its environment and trying out different actions, and receives rewards correspondingly. It should also sufficiently explore the possible action space, while still eventually converging to an optimal path of action. Furthermore, actions taken at a certain point in time usually affect the environment and future reward. The paradigms of trial-and-error, exploration-exploitation, and delayed gratification are, therefore, central to RL. Learning from experience in this fashion allows RL to engage with interactive and unfamiliar problems where examples of optimal behaviour might not be available, and which are, therefore, not readily addressable by supervised learning techniques.

This section elucidates the concepts of RL and the SAC algorithm. It introduces the principles of RL, outlines the SAC algorithm, and discusses the mathematical underpinnings of SAC, including policy iteration, entropy maximisation, and the modified Bellman backup operator. The section culminates by emphasising the scalability and stability of SAC, its ability to address high sample complexity and convergence brittleness, and its effectiveness in balancing exploration and exploitation.

Soft Actor-Critic

Introduced by Haarnoja et al. [14], the SAC is an RL algorithm that addresses the challenges typically faced by model-free RL, such as high sample complexity and convergence brittleness. High sample complexity implies a large computational cost, and convergence brittleness necessitates tedious hyperparameter tuning to ensure stable learning and convergence. In contrast to most RL algorithms, SAC aims to maximise not only the expected reward, R, but also the action-selection entropy. This is achieved by taking maximally random actions off-policy, while still optimising the policy for the given task. These random, albeit high-entropy actions, encourage diverse behaviour [30,31] and promote exploration of the action space. High-entropy actions provide resilience against estimation errors [32], as the policy assigns similar probabilities to actions with similar values to prevent the policy from continually reselecting the same action. Moreover, SAC can handle continuous, high-dimensional state and action spaces, which often pose problems for convergence stability [33].

The key constituents of SAC include an actor for the policy function, and a critic for the value function. Furthermore, off-policy learning increases efficiency by incorporating previously collected data, and entropy maximisation enhances exploration and reduces brittleness. In adherence to the actor-critic formulation, SAC performs policy iteration by cycling between policy evaluation, which calculates the value function v_π of a policy π, and policy improvement, where the value function is used to improve the policy [13]. Policy iteration for SAC has a convergence proof [14]. Entropy maximisation ensures that random actions are chosen using

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} [r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot \mid \mathbf{s}_t))] \tag{15}$$

where $\mathbf{s}_t$ represents the current state in state space $\mathcal{S}$, and $\mathbf{a}_t$ is the current action in action space $\mathcal{A}$. The expectation is taken over the state transition probability, $\rho_\pi(\mathbf{s}_t)$. The reward function is denoted by r. This formulation yields a Markov decision process (MDP) defined as $(\mathcal{S}, \mathcal{A}, p, r)$ [13]. The temperature parameter α balances the weighting between entropy and reward, where a lower α results in less randomness in the optimal policy v^* and, consequently, less exploration.

The Q-value represents the expected reward for a certain state-action pair, and is obtained by iteratively calculating the modified Bellman backup operator using

$$\mathcal{T}^{\pi}Q(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p}[V(\mathbf{s}_{t+1})] \tag{16}$$

from any starting function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The discount factor, γ, determines how myopic the agent is. For a small γ, the agent optimizes for immediate reward at the expense of potentially more reward later. Conversely, a higher γ incentivises 'delayed gratification', where the agent might sacrifice reward at the current time-step if such action ensures increased reward at a later stage. After calculation of Equation (16), the value of the policy is calculated as

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi}[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t \mid \mathbf{s}_t)] \tag{17}$$

During policy improvement, π is updated using the Kullback–Leibler divergence [34] projection to constrain the policy to the set of tractable policies, Π, according to

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'(\cdot \mid \mathbf{s}_t) \middle\| \frac{\exp(Q^{\pi_{\text{old}}}(\mathbf{s}_t, \cdot))}{Z^{\pi_{\text{old}}}(\mathbf{s}_t)}\right) \tag{18}$$

where $Z^{\pi_{\text{old}}}(\mathbf{s}_t)$ is the partition function which normalises the distribution.

In order to extend Equations (16)–(18) to large, continuous domains, the policy π must be approximated to the tractable policy π_ϕ, the Q-function Q to the "soft" Q-function Q_θ, and the value function V to an approximate state value function V_ψ. Parameters ϕ, θ, and ψ, respectively, govern these approximations. Also, since the complete, iterative evaluation of Equation (16) quickly becomes intractable, both approximations are updated alternatingly, rather than executing both evaluation and improvement to completion. The soft Q-function is, therefore, minimised with respect to the squared residual error,

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}}\left[\frac{1}{2}\left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi\phi}[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t)]\right)^2\right] \tag{19}$$

where the error is the difference between the approximated value V_ψ, and the expectation of Q_θ in conjunction with the entropy of π_ϕ. The expectation is taken over all of $\mathcal{D}$, the distribution of all previous states and actions.

The SAC algorithm is a powerful tool for solving RL problems, demonstrating high efficiency and robustness. The actor-critic architecture allows for more efficient and effective learning by combining both value-based and policy-based methods. Entropy maximisation encourages diverse behaviour and exploration, reducing the brittleness commonly associated with model-free RL methods. The use of off-policy learning enhances sample efficiency, making better use of previously collected data. Overall, SAC offers a scalable and stable approach to RL, particularly in high-dimensional and continuous state and action spaces, and strikes a balance between exploration and exploitation.

2.7. Reinforcement Learning as a Hyper-Heuristic

Selection hyper-heuristics are control mechanisms that identify the most suitable optimisation technique over time, without complete knowledge of the problem domain [35]. Selection hyper-heuristics can function as operators which choose from a collection of population-based optimisation methods like PSO, which run concurrently. This approach is known as the heterogeneous meta-hyper-heuristic (HMHH) framework [36–38]. Although the focus of this paper is not the design of selection hyper-heuristics, HMHH warrants mention as it has seen the application of RL to population-based optimisation methods. Note, however, that in this context, RL does not refer to an agent being trained, but rather to the fact that a feedback signal is used to change the selection of the optimisation method.

This section examines the application of RL in selection hyper-heuristics within the HMHH framework. Two RL-inspired approaches, frequency improvement reinforcement learning Selection (HH_ReinfFr) and objective function proportional reinforcement learning selection (HH_ReinfPr), are discussed.

2.7.1. Frequency Improvement Reinforcement Learning Selection

HH_ReinfFr [35] is a hyper-heuristic approach inspired by the works of [39,40]. In this method, each heuristic h_m has a rank score r_m, initially set to zero. The heuristics are rewarded or punished based on the frequency with which they improve the quality of assigned candidate solutions. The change in rank for a heuristic h_m is calculated as $\Delta r_m(t) = \chi_m(t)$, where $\chi_m(t)$ represents the frequency score according to

$$\chi_m(t) = \sum_{m=1}^{n_t} \sum_{i=1}^{n_m} \begin{cases} +1 & \text{if } f_i \text{ improved} \\ -1 & \text{if } f_i \text{ remained the same} \\ -1 & \text{if } f_i \text{ worsened} \end{cases} \tag{20}$$

where f_i is the quality of candidate solution i, n_m is the number of candidate solutions assigned to heuristic h_m, and n_t is the number of iterations over which the calculation is performed. The range of $\chi_m(t)$ is $[-(n_s \times n_t), (n_s \times n_t)]$. Rank update happens according to $r_m(t) = r_m(t - n_t) + \Delta r_m(t)$. Candidate solutions are assigned to the highest-ranked heuristic, following the approach by [40], in effect, blacklisting all other heuristics. The values in $\Theta(t)$ are assigned as $P_m = 1$ for $m = r$ for the highest-ranked heuristic $h_r \in H$, while $P_m = 0$ for $m \neq r$. Rank ties are resolved randomly, and the same $\Theta(t)$ is employed for assigning all candidate solutions.

2.7.2. Objective Function Proportional Reinforcement Learning Selection

HH_ReinfPr [35] is another hyper-heuristic approach. While similar to HH_ReinfFr, the difference lies in the adjustment of ranks, which is based on the mean change in quality of candidate solutions. The mean change in solution quality is calculated using

$$\phi_m(t) = \frac{\sum_{i=1}^{n_m} (f_i(t) - f_i(t - n_t))}{n_m} \tag{21}$$

The change in rank $\Delta r_m(t)$ for each heuristic h_m is calculated as

$$\Delta r_m(t) = \begin{cases} +1 & \text{if } \phi_m(t) > 0 \\ -1 & \text{if } \phi_m(t) = 0 \\ -1 & \text{if } \phi_m(t) < 0 \end{cases}$$

Ranks are updated using the formula $r_m(t) = r_m(t - n_t) + \Delta r_m(t)$. Initial ranks for all heuristics are $r_m = r_{\min} = 0$. As for HH_ReinfFr, candidate solutions are assigned to the highest-ranked heuristic in accordance with the approach by [40]. $\Theta(t)$ probabilities are set as $P_m = 1$ for $m = r$ in the case of the highest-ranked heuristic $h_r \in H$, while $P_m = 0$ for $m \neq r$. Rank ties are broken at random, and the same $\Theta(t)$ is used for assigning all candidate solutions. The difference between HH_ReinfFr and HH_ReinfPr is that the former rewards heuristics based on the frequency with which they improve solution quality, while the latter rewards heuristics based on the mean change in quality of candidate solutions assigned to them.

3. Design of Soft Actor-Critic Self-Adaptive Mechanism

This section details the design of the SAC-SAPSO algorithm, which uses the SAC to dynamically adjust the PSO CPs. Initially, the section motivates why RL is ideally suited, and a suitable RL algorithm is chosen. Thereafter, the SAC-SAPSO architecture is described, including design choices pertaining to the RL environment, agent, and reward function.

3.1. Motivation for Reinforcement Learning for Self-Adaptation

Development of a self-adaptive PSO algorithm essentially amounts to the adjustment of CP values, based on the behaviour of the particles, so as to improve the solution quality. This can be framed as an RL problem, where an agent finds a policy to govern its actions (choosing new CP values), based on observations (behaviour of the particles), in order to maximise a reward signal (solution quality). Using RL for SAPSO is somewhat related to UAPSO-A [29], because the UAPSO-A learning automata can be considered as simple RL agents which receive reward if the selected CP values improved solution quality. However, UAPSO-A is subject to several limitations:

- UAPSO-A does not consider particle characteristics such as velocity or position to inform CP updates.
- Orthogonal automata are used for each CP, which precludes the decision-making mechanism from considering the other CP values when updating the current CP.
- The UAPSO-A mechanism makes use of simplistic probability-based decision-making, which assumes previously successful CP values to be successful in the future. This assumption is not necessarily true, given deceptive optimisation problems with changing function landscape characteristics.
- UAPSO-A can only select CP values from a discrete set, disallowing fine-grained adjustment.

Moreover, RL has been used to statically pre-select hyperparameters or to tune hyperparameters [15–18]. However, no examples were found where RL is used to dynamically adjust hyperparameters during execution of the algorithm. Algorithms such as HH_ReinfFr and HH_ReinfPr (discussed in Sections 2.7.1 and 2.7.2) also use RL, but they select amongst different heuristics, which also differs greatly from dynamic adjustment of PSO CPs during the search process.

A significant advantage of using RL for self-adaptation is that an RL agent can automatically discover the best policy. Most SAPSO approaches are based on inductive priors, i.e., the authors believe that adapting the CPs in a certain way is more likely to lead to a good outcome. For example, PSO-TVAC (see Section 2.2) is based on the assumption that decreasing particle inertia will lead to improved performance. While this may well be the case, it is difficult to reason a priori about the behaviour of the PSO swarm. For example, there is complex interplay between the fitness landscape, the CP configuration, and the particle positions and solutions. Correlations between these factors cannot easily be captured by a simple heuristic, but can be learned by an RL agent, which can then adjust the CP configuration accordingly. Also, if performed manually, only a few approaches toward SAPSO can be tried, whereas an RL agent explores millions of policies.

Using supervised neural networks (NNs) to adapt PSO CPs has also been considered. For example, the input to the NN could be information about the particles, such as fitness landscape characteristics at current positions, and the output could be the CP configuration predicted for the next time-step. This approach is inferior to RL for the following reasons:

- Labelling vs. Interacting: In order to train the NN, a labelled dataset is required. Since the NN has to predict the CPs, supervised learning requires knowledge of the "correct" CPs in order for the NN to train, which is, of course, not known in advance. RL bypasses this issue, as the agent interacts with the environment autonomously, and learns from experience. In this case, the agent is the SAC, and the environment is the PSO search process, which provides information such as particle velocities and solutions to the agent.
- Exploration vs. Exploitation: Just as the particle swarm should ideally maintain a suitable balance between exploration and exploitation, this balance should also be present in the CP space. NNs are not capable of exploring the CP space, but merely learn from labels. Conversely, the exploration-exploitation paradigm lies at the heart of the SAC formulation, as discussed in Section 2.6. The SAC agent explores the CP

space by taking random actions, and exploits the CP space by taking actions which have been shown to be successful in the past.

To conclude, RL offers a promising avenue for self-adaptation in PSO algorithms by dynamically adjusting CP values during the execution of the algorithm. RL overcomes several limitations of existing methods such as UAPSO-A and supervised neural networks by enabling a more fine-grained, informed, and adaptive decision-making process. By interacting with the environment and learning from experience, an RL agent can discover optimal policies and maintain a balance between exploration and exploitation in the CP space, leading to improved performance and solution quality in complex optimisation problems.

3.2. Reinforcement Learning Algorithm Selection

A plethora of RL algorithms exist, some more suitable than others towards SAPSO. The following criteria are considered:

- **Model-free, not model-based:** Since there is no preexisting model of how the CP configuration should be adapted (if there was, the problem would be solved already), the agent has to learn in a model-free way.
- **Off-policy, not on-policy:** Exploration of the CP space is aided if the agent is permitted to take actions off-policy. If the agent only takes actions on-policy, the only possible actions are those which have been successful in the past, increasing the likelihood of becoming stuck in a local optimum.
- **Continuous action space, not discrete:** The possible CP values which can be selected form a continuous range and the policy should, therefore, have a continuous action space. Contrast this with the UAPSO-A algorithm in Section 2.5.2, which outputs a range of discrete values. Discretisation adds complexity because the agent chooses between n different values on every CP update. Nonetheless, it has less flexibility than a continuous action space because it is still limited to those n specific values.
- **Continuous state space, not discrete:** All variables in the PSO formulation (i.e., particle positions, velocities, etc.) are continuous floating-point values. Since these are the variables which the agent observes, the agent should have a continuous state (observation) space.

A comparison of RL algorithms is given in Table 1, from which deep deterministic policy gradient (DDPG), twin delayed DDPG (TD3), and SAC are eligible according to the aforementioned criteria. Since TD3 and DDPG are more prone to hyperparameter instability [14], they are eliminated for the purposes of this study.

Table 1. Comparison of RL algorithms. (Discr. = discrete action space, Cont. = continuous action space).

Algorithm	**Definition**	**Action**	**State**	**Model**	**Policy**
MC [41]	Every-Visit Monte Carlo	Discr.	Discr.	Model free	Either
SARSA [42]	State-action-reward-state-action	Discr.	Discr.	Model free	On-policy
Q-learning [43]	State-action-reward-state	Discr.	Discr.	Model free	Off-policy
DQN [44]	Deep Q-Network	Discr.	Cont.	Model free	Off-policy
A3C [45]	Asynchronous Advantage Actor-Critic	Cont.	Cont.	Model free	On-policy
TRPO [46]	Trust Region Policy Optimisation	Cont.	Cont.	Model free	On-policy
PPO [47]	Proximal Policy Optimisation	Cont.	Cont.	Model free	On-policy
DDPG [48]	Deep Deterministic Policy Gradient	Cont.	Cont.	Model free	Off-policy
TD3 [49]	Twin Delayed Deep Deterministic Policy Gradient	Cont.	Cont.	Model free	Off-policy
SAC [14]	Soft Actor-Critic	Cont.	Cont.	Model free	Off-policy

Furthermore, in cases where the RL environment is not computationally expensive to simulate or execute, it is often preferable to use an algorithm with lower sample efficiency and to run it for more iterations. However, since evaluation of the environment (i.e., the PSO search) amounts to computing velocities and positions for n_s particles in n_d dimensions, this makes for a rather complex environment. For this reason, an algorithm with higher sample efficiency allows for the environment to be evaluated less often. SAC fits the aforementioned criteria, because it has higher-than-normal sample efficiency. It is also robust to convergence brittleness, as discussed in Section 2.6, and is, therefore, the algorithm of choice for the development of the RL-based SAPSO.

3.3. SAC-SAPSO Architecture

The architecture of SAC-SAPSO consists of the PSO and the SAC agent, which are implemented as described in Sections 2.1 and 2.6, respectively. Interaction between PSO and SAC lies in that the SAC agent observes the PSO search process, which is the environment. The SAC agent then takes actions which influence the PSO environment.

This section presents the SAC-SAPSO architecture, consisting of the PSO and a SAC agent. The interaction between these components is explained, as well as the PSO environment details, the SAC agent's actions, and the formulation of the SAC-SAPSO reward signal. Detailed discussion covers topics like PSO metrics, normalisation of velocities, SAC agent's policy, and benefits and drawbacks of absolute and relative reward signals.

The SAC-SAPSO is summarised in Algorithm 1.

Algorithm 1: SAC-SAPSO Algorithm

```
1  Create and initialise an $n_x$-dimensional swarm, $S$;
2  Create and initialise the SAC RL agent and neural network architecture;
3  Predict initial values for the PSO control parameter values, $w, c_1$ and $c_2$, using the
     RL agent;
4  Initialise iterations, $t = 0$;
5  Set the RL agent observation frequence, $n_t$;
6  repeat
7      $t = t + 1$;
8      for each particle $i = 1, \ldots, S.n_s$ do
9          if $f(S.\mathbf{x}_i) < f(S.\mathbf{y}_i)$ then
10             $S.\mathbf{y}_i = S.\mathbf{x}_i$;
11         if $f(S.\mathbf{y}_i) < f(S.\hat{\mathbf{y}})$ then
12             $S.\hat{\mathbf{y}} = S.\mathbf{y}_i$;
13     for each particle $i = 1, \ldots, S.n_s$ do
14         update the velocity;
15         update the position;
16     if $t$ mod $n_t$ = 0 then
17         Predict new control parameter values for $w, c_1$ and $c_2$ using the RL agent;
18         Calculate the reward signal;
19         Update the RL agent policy (i.e., adjust the neural network predictive
             model);
20 until stopping condition is true;
21 Return $S.\hat{\mathbf{y}}$ as the solution;
```

3.3.1. PSO as the Environment

The PSO search process serves as the environment for the SAC agent, which observes particle velocities, percentage stable particles, percentage infeasible particles, and the percentage completion of the search at any given time-step. See Section 4.1 for a discussion of these metrics, except for percentage completion, which is defined as the number of the current time-steps divided by the maximum number of time-steps. The observation space thus has a dimensionality of $n_s + 3$, where n_s is the number of particles in the swarm. This is because particle velocities are observed for each particle, while the other metrics are calculated for the entire swarm. To ensure a unitary observation space for the SAC agent, particle velocities are normalised and scaled according to the transformation,

$$\mathbf{v}_{\text{norm}} = \tanh\left(\frac{2}{\mathbf{u}-\boldsymbol{\ell}} \cdot \left(\mathbf{v} - \frac{\boldsymbol{\ell}+\mathbf{u}}{2}\right)\right) \tag{22}$$

where $\mathbf{v}$ is the velocity vector of the particle, $\boldsymbol{\ell}$ is the lower bound of the feasible search space, and $\mathbf{u}$ is the upper bound of the feasible search space. Equation (22) first normalises particle velocities by the boundaries of the feasible search space. This ensures that a velocity of 1 corresponds to traversal of the entire search space in one time-step. However, since velocity explosion [9,25] is a common problem in PSO, particle velocities can assume extremely large values. To still retain a unitary observation space, the normalised velocity vector is then squashed using the hyperbolic tangent function to the range of $[-1, 1]$. Intuitively, a particle with a velocity which exploded to 10^5 has just as little chance of finding a good solution in feasible space as does a particle with a velocity of 10^{10}. The hyperbolic tangent function accounts for this, with all such large velocities being squashed to approximately 1.

3.3.2. SAC as the Agent

The SAC agent takes actions according to a policy, which governs the adjustment of the PSO CPs during the search process. Observation of the PSO environment is performed every n_t time-steps. Upon observing the environment, the agent outputs as actions the continuous CP values of ω, c_1, and c_2 at that time-step. Since all particles are given the same CP values, the action space has a dimensionality of 3. These values are then used to update the PSO CPs according to the velocity update rule in Equation (1).

Updating the CPs on every time-step (setting $n_t = 1$) is inefficient, because the search characteristics do not change sufficiently during one time-step to necessitate a new CP configuration. A larger value for n_t can be used, but introduces an additional hyperparameter. Alternatively, n_t can form part of the action space, thus allowing the agent to autonomously learn an optimal value for n_t. The latter option has as an advantage that the agent can learn to adjust the observation-action frequency according to the current state of the search process, potentially improving the search performance. The three options are, therefore, $n_y = 1$ to adapt on every time-step, $n_t = t^{'}$ to adapt every $t^{'}$ time-steps, and $n_t \in \mathcal{A}$ (the action space).

3.3.3. SAC-SAPSO Reward Signal

The reward signal informs an RL agent whether its actions achieve the desired outcome. The reward acts as a feedback mechanism and guides the agent towards learning the optimal policy, which maximizes the expected cumulative reward over time [13]. Proper reward shaping is crucial to align the goal of the SAC agent with that of the PSO, namely, to ensure that CP adaptation improves search performance. Since the minima and maxima of the training function set are known, it is straightforward to relate the global best solution to an absolute normalised reward signal according to

$$r_{\text{abs}}(t) = \frac{y_f(t-1) - y_f(t)}{y_{max} - y_{min}} \tag{23}$$

where y_f is the global best solution at time-step t. The term 'absolute' indicates that the normalisation is performed using the absolute minimum and maximum of each function in the training function set. Note that the generic formula for absolute change is usually defined as

$$\delta_{\text{abs}} = \frac{y_{new} - y_{old}}{y_{max} - y_{min}} \tag{24}$$

However, in Equation (23), the numerator is reverted, because a decreasing global best solution implies an increasing reward. Normalisation is necessary, because the functions lie on vastly different magnitude ranges, and ensures a direct relationship between reward maximisation and improvement of the global best solution found. Equation (23) has the reward landscape shown in Figure 1a.

The reason for computing the numerator of $r_{\text{abs}}(t)$ with respect to $y_f(t-1)$ instead of y_{max} is to ensure that the maximum possible reward for a given function, which aggregates during the PSO search of that function, equals 1. The agent thus receives a cumulative reward of 1 only if the PSO manages to find y_{min} during the search process. Similarly, a cumulative reward of 0.5 indicates that the PSO only found a global best solution halfway between y_{max} and y_{min}. If the reward was calculated with respect to y_{max}, this would provide less incentive towards finding an optimal solution. For example, if a global best solution at a given time-step produced a reward of 0.5, the agent would continue to receive a reward of at least 0.5 for all subsequent time-steps, even if the global best solution stagnated there.

The reward formulation of Equation (23) is also preferable to that of UAPSO-A (Section 2.5.2), where a constant amount of reward is generated if the objective function value improved, irrespective of the degree of improvement. This UAPSO-A reward formulation could incentivise the agent into optimising for small, incremental steps, instead of just solving the function optimally.

Equation (23) evidently has the problem that it can only be used if the minimum and maximum of the objective function are known. If the minimum is known, the objective function has, of course, already been solved. To circumvent this, Equation (23) is modified to obtain a relative reward signal, defined as

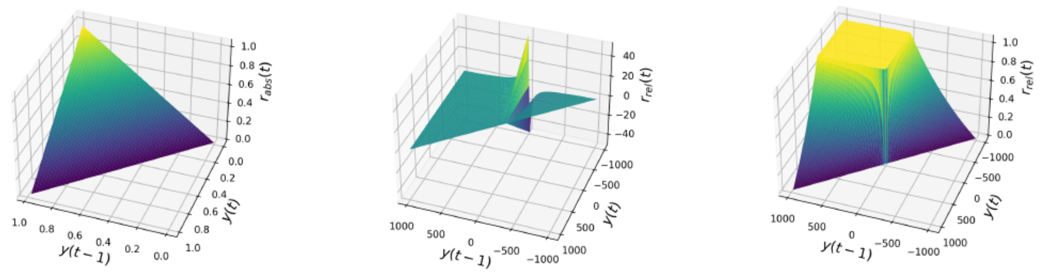
$$r_{\text{rel}}(t) = \begin{cases} 0 & \text{if } y_f(t) = y_f(t-1) \\ 2 \cdot \frac{y_f(t-1)+\beta-y_f(t)+\beta}{y_f(t-1)+\beta} & \text{if } y_f(t) > 0, y_f(t-1) > 0 \\ 2 \cdot \frac{y_f(t-1)+2\beta-y_f(t)+2\beta}{y_f(t-1)+2\beta} & \text{if } y_f(t) < 0, y_f(t-1) < 0 \\ 1 & \text{otherwise} \end{cases} \tag{25}$$

where $\beta = |y_f(t-1)| + |y_f(t)|$. The reward, $r_{\text{rel}}(t)$, is, therefore, always only calculated as the improvement in solution quality with respect to the previous time-step, and never makes use of the absolute maximum and minimum of a function. However, functions can take on any range; that is, y_{min} and y_{max} can both be positive, both be negative, or negative and positive, respectively, and can be arbitrarily large or small. This precludes simply using the general formula for relative change (with an inverted numerator),

$$\delta_{\text{rel}} = \frac{y_{\text{old}} - y_{\text{new}}}{y_{\text{old}}} \tag{26}$$

which only works if all values have the same sign, and does not provide unitary normalisation. If not, Equation (26) results in a jump discontinuity as soon as y_{new} changes from positive to negative during the PSO search. Also, when both y_{old} and y_{new} are negative, the reward is distorted and inverted. The aforementioned problems are clear in Figure 1c.

Equation (25) addresses these problems by shifting y_{old} and y_{new} by 2β if both are negative, β if both are positive, and scales by 2 for both. The shift is calculated in this way to ensure that after shifting, both y_{old} and y_{new} are positive and non-zero. Equation (25) also places a fixed limit of $r_{\text{rel}_{\text{max}}} = 1$ for the rare time-steps where the global best solution transitions from positive to negative. The scaling factor of 2 ensures continuity between the four possible cases, with the resulting reward landscape shown in Figure 1b. Note that the objective function value y in Figure 1b,c has not been normalised, with $[-3,3]$ selected arbitrarily, because the range of y can take any value. Conversely, y is always normalised before calculating Equation (23) to obtain the landscape in Figure 1a, which is why the range is $[0,1]$. Furthermore, while the area in Figure 1b, where $r_{\text{rel}} = 1$, appears to be very large, the reality is that it only occurs at time-steps where $y(t-1) > 0$ and $y(t) < 0$. These time-steps are rare, because they only occur one or zero times during the PSO search.



(**a**) (**b**) (**c**)

Figure 1. Comparison of reward landscapes. (**a**) Absolute reward (Equation (23)). (**b**) Relative reward (Equation (25)). (**c**) Relative change (Equation (26)).

4. Experimental Procedure

This section elaborates on the experimental procedure, focusing on evaluation metrics and specifics regarding the PSO implementation. Following this, the details of the algorithm's implementation are presented. This includes the nature of RL, the role of training and testing sets, and the specifics of conducting an experiment. Lastly, the section delves into the hyperparameters used during the training phase of the SAC algorithm, underscoring their importance in the overall effectiveness and functionality of the SAC-SAPSO algorithm. As various aspects of the algorithm are assessed, a 'run' is defined as a single SAC-SAPSO execution on a single function, while an 'experiment' denotes $r = 30$ independent runs for each function in the entire function set.

4.1. Evaluation Metrics

Multiple metrics are utilised to examine the performance and characteristics of the SAC-SAPSO:

1. **Normalised global best solution quality.** The objective function value, or solution quality at the global best positions, signifies how well an algorithm solves a given optimisation problem. Because functions differ in their ranges, normalised global best solution quality is employed. Once all experiments have concluded, the highest and lowest global best solutions discovered throughout all experiments and runs are used to normalise global best solutions to [0, 1] for each experiment. The scale used for normalisation is chosen at will, but unitary scaling leads to easily comprehensible results, where lower values represent better solutions.
2. **Average swarm diversity** offers insights into the balance of exploration and exploitation. Diversity is computed using [50]

$$\mathcal{D} = \frac{1}{n_s}\sum_{i=1}^{n_s}\sqrt{\sum_{j=1}^{n_x}\left(x_{ij} - \bar{x}_j\right)^2} \tag{27}$$

   with the swarm center located at

$$x_j = \frac{\sum_{i=1}^{n_s} x_{ij}}{n_s} \tag{28}$$

   where n_s represents the number of particles in the swarm, and n_x refers to the number of dimensions.
3. **Percentage of particles in infeasible space.** A particle is considered to be in infeasible space if it breaches the boundaries of the feasible search space in at least one dimension. Infeasible particles should not be taken into account when updating the best positions found, so as not to steer the search away from feasible space [9].

4. **Percentage of particles that are stable.** Poli's stability condition (given in Section 2.3 in Equation (4)) determines whether a particle has convergent CP configurations, and is considered stable [51].
5. **Average particle velocity** denotes the average step sizes. In order for the search process to converge, step sizes have to decrease. However, in order to not become trapped in a local minimum, step sizes should not become too small too quickly. The average particle velocity is calculated using [7]

$$\Delta(t+1) = \frac{1}{n_s}\sum_{i=1}^{n_s}\|\mathbf{x}_i(t+1) - \mathbf{x}_i(t)\|. \tag{29}$$

4.2. Implementation

Unlike supervised learning methods, which require separate training and testing sets to learn and evaluate the performance of a model, RL algorithms are trained on the environment for which they are used. The reason behind this difference lies in the nature of the RL learning process. In supervised learning, the model learns from a fixed set of labelled examples, and the testing set is used to evaluate the generalisation ability of the model to unseen data [52]. Conversely, RL agents take actions based on observations to solve environments [13]. Therefore, for each experiment (i.e., algorithmic aspect under investigation), an agent is trained and $r = 30$ independent evaluation runs. This is performed using the set which contains 45 30-dimensional minimisation functions provided in Appendix A.

However, since the SAC-SAPSO algorithm must definitely generalise, the training function set cannot also represent algorithmic performance without bias. Thus, a test set of seven additional functions is used, also detailed in Appendix A. This ensures an 85% training, 15% testing split for honest assessment of performance. The test functions have specifically been chosen to be more complex than the training set in order to demonstrate performance differences between algorithms more clearly.

The inertia weight PSO [19] is employed (described by Equations (1) and (2)), and the neighbourhood for each particle encompasses the entire swarm. The objective function value is only computed for particles situated within feasible space. Particles that fall outside the viable space are assigned infinite objective values, which, under the assumption of minimisation, disqualifies those particles from global best position updates. The swarm size is $n_s = 30$, with each function having a dimensional space of $n_x = 30$, with a maximum of $t_{max} = 5000$ iterations for each run.

The average and standard deviation for all plots and scores are calculated across all functions and runs within a specific experiment, resulting in $(30 \times 45) + (30 \times 7) = 1560$ runs for each experiment. Given the various algorithmic components examined, a total of 15 experiments are conducted, which translates to 23,400 runs.

4.3. SAC Hyperparameters

The SAC training hyperparameters are shown in Table 2. Note that the SAC training hyperparameters differ from PSO CPs because they are only used during the training of the SAC policy, whereas the CPs are dynamically adjusted during the PSO search to improve performance. Moreover, since the impact of these hyperparameters on the SAC agent is well understood and documented [31], it is possible to reason a priori about the values that should be used. Default values are also provided in the SAC documentation (https://spinningup.openai.com/en/latest/, accessed on 1 November 2023). Finally, these are all training hyperparameters. Once a policy is learned which successfully adapts the PSO CPs on the training set, these hyperparameters are discarded, and have no bearing on the functioning of the SAPSO algorithm anymore. The SAC hyperparameters are, therefore, not counter to the aim of SAPSO.

The discount factor γ governs the trade-off between immediate and future rewards, where a low γ will result in the agent being myopic, and a high γ will result in the agent being far-sighted. For infinite-horizon RL problems, γ usually needs to be adjusted carefully. However, for episodic RL problems such as the PSO search, $\gamma = 1$ is recommended [13].

The reward scale factor f_r can be used to normalise the reward signal. However, the reward formulation defined in Equation (25) is already normalised to the range $[0, 1]$. Therefore, the reward is not scaled here, i.e., f_r is not used. For completeness, this is equivalent to setting $f_r = 1$.

Actor and critic layer sizes are set to 256 each, the default values recommended by the SAC documentation. Barring the case of too small layer sizes, which could render the policy too simple to learn complex PSO dynamics, these hyperparameters are not expected to have a significant impact on the performance of the learned policy.

Two value (critic) networks and one policy (actor) network are used by the SAC [14]. The target smoothing coefficient τ controls the rate at which these networks are updated using

$$\mathbf{w}_{\text{target}} \leftarrow \tau \cdot \mathbf{w}_{\text{critic}} + (1 - \tau) \cdot \mathbf{w}_{\text{target}} \tag{30}$$

where $\mathbf{w}_{\text{target}}$ represents the weights of the target network, and $\mathbf{w}_{\text{critic}}$ denotes the weights of the critic network; τ lies between 0 and 1, where a smaller τ corresponds to slower updates, resulting in more stable value estimates but potentially slower learning. A larger τ leads to faster updates, which may cause higher variance in value estimates and less stable learning. The default, small value of $\tau = 0.005$ is used here, as recommended by the SAC documentation, because slower learning during training does not have a bearing on the actual usage of the SAC-SAPSO.

The learning rate α is set to a small, default value of $\alpha = 0.0003$. Similar reasoning as for τ applies here.

Replay buffers are used to store past experiences, which are sampled from during training. The main concern regarding the replay buffer size is potential memory constraints during training. A large, default value of 10^6 is chosen.

The number of training steps are set to 2×10^5. Because the training is actually stopped upon stagnation in performance of the learned policy, this value has no bearing on performance and is chosen as an arbitrary, large value.

All the hyperparameters in Table 2 are training parameters. This disjoint between training and execution ensures no runtime CPs, except potentially n_t. However, n_t can be set to form part of the SAC action space $\mathcal{A}$, which completely automates this parameter also, as discussed in Section 3.3.2. Such a training-execution disjoint may well be the only way to adapt all PSO CPs, corroborated by the fact that all approaches to SAPSO end up introducing more runtime CPs in an attempt to adapt ω, c_1 and c_2, as shown by [7,8].

Table 2. SAC hyperparameters.

Discount factor γ	1	Target smoothing coefficient τ	0.005
Reward scale factor f_r	1	Learning rate α	0.0001
Actor layer size	256	Replay buffer size	10^6
Critic layer size	256	Training steps	2×10^5

5. Results

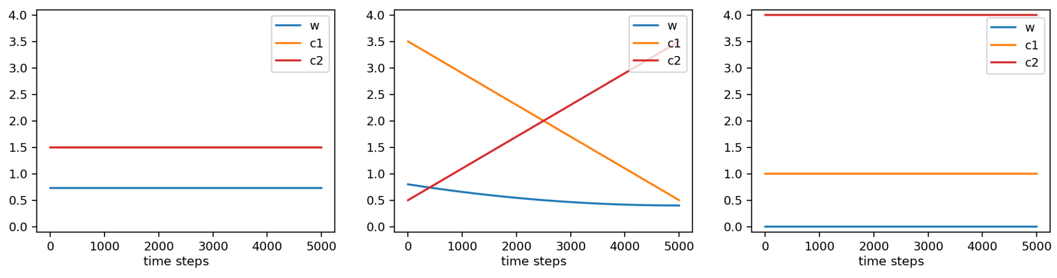
This section details the algorithmic aspects of different experiments, together with their respective results. Four performance baselines are presented together with their comparative outcomes. The final part explores the functionality of the SAC-SAPSO algorithm and its performance implications, highlighting how it outperforms the established baselines. Some plots contain gaps, such as Figure 11e, which occur when particles drift far enough out of the search space to cause numerical overflow in the swarm diversity calculation from Equation (27). Plot lines represent performance averages, and shaded areas the standard deviation, calculated over all functions and runs for the experiment at hand, as discussed in Section 4.2.

5.1. Performance Baselines

Four baselines are defined in this section, and their performance is presented in Table 3 and illustrated in Figures 2–6. The first baseline (baseline_constant) employs the constant CP setup outlined in Section 2.2, and the second one (baseline_timevariant) applies the time-variant configuration described by Equation (3). PSO-TVAC (see Section 2.2) is used as the third baseline (baseline_tvac), and the final baseline (baseline_random) randomly samples CPs which adhere to Poli’s convergence criterion in Equation (4) at each time step, for every particle in the swarm [53,54].

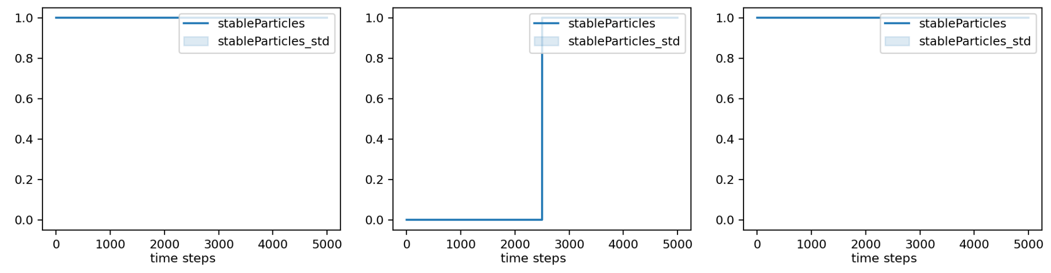
Table 3. Baseline normalised global best solution quality.

	Training		**Testing**	
	μ	σ	μ	σ
baseline_tvac	0.0883	0.1894	0.2926	0.2037
baseline_timevariant	0.0888	0.1991	0.3287	0.3052
baseline_constant	0.0893	0.1856	0.3802	0.2822
baseline_random	0.1044	0.2032	0.3812	0.3125



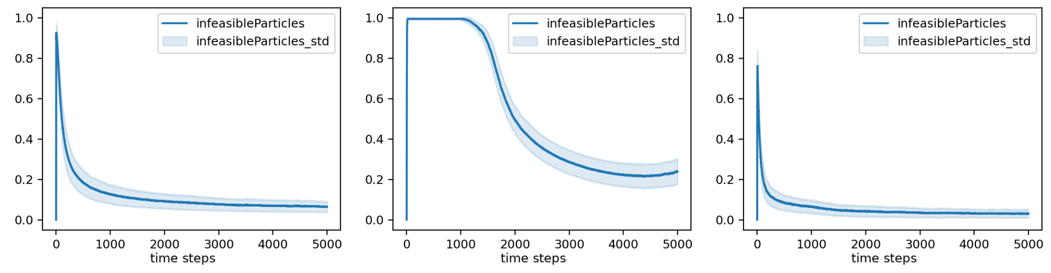
(**a**) (**b**) (**c**)

Figure 2. Control parameter values for baseline. (**a**) Constant CP (note that plots for c_1 and c_2 coincides); (**b**) Time-variant CP; (**c**) Random CP.



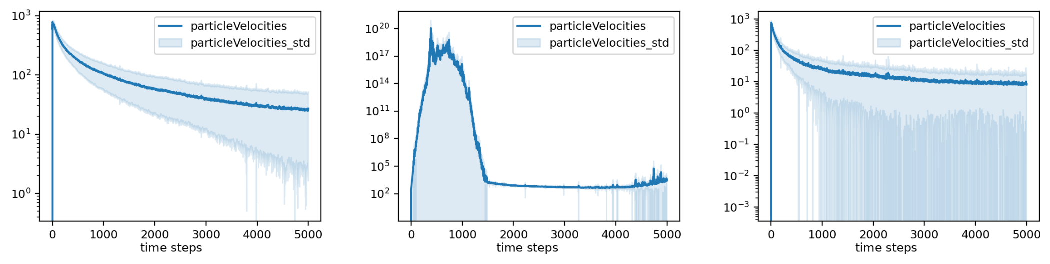
(**a**) Constant CP (**b**) Time-variant CP (**c**) Random CP

Figure 3. Stable particles for baseline.



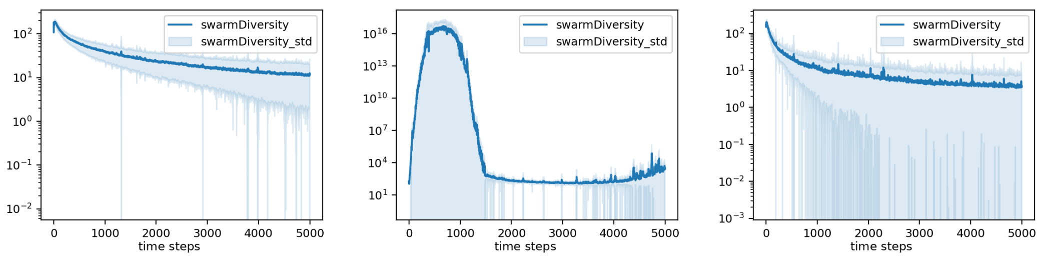
(**a**) Constant CP (**b**) Time-variant CP (**c**) Random CP

Figure 4. Infeasible particles for baseline.



(**a**) Constant CP (**b**) Time-variant CP (**c**) Random CP

Figure 5. Particle velocity vectors for baseline.



(**a**) Constant CP (**b**) Time-variant CP (**c**) Random CP

Figure 6. Swarm diversity for baseline.

5.2. *Soft Actor-Critic Control Parameter Adaptation*

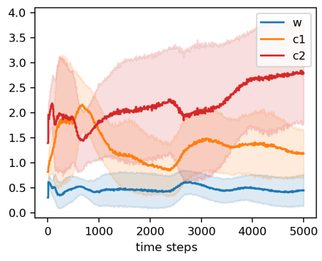
The SAC-SAPSO algorithm is evaluated for various values of n_t (sac_$n_t = k$), in addition to an experiment where n_t is set by the SAC algorithm itself (sac_n_t = auto), with results given in Table 4. Figure 7 shows how the CP values are adapted by the SAC-SAPSO algorithm throughout the search process, and Figure 8 shows the number of stable particles in the swarm. On average, about half of the particles in the swarm have stable (convergent) CP configurations, with generally less than 40% of particles residing in infeasible space, as seen in Figure 9. Nonetheless, Figure 10 clearly shows that some particles in the swarm have exploding velocities, which results in the exploded swarm diversity in Figure 11. In some cases, the swarm diversity becomes big enough to cause numerical overflow, indicated by gaps, such as in Figure 11e.

For sac_n_t = auto, when considering Figures 7f and 8f jointly, it is interesting to note how the agent which adapts n_t autonomously ultimately decides to keep exploration (c_1, ω) high (with an entirely stable CP swarm!) for approximately the first 1000 time-steps. Thereafter, exploration and exploitation are promptly switched around by lowering c_1 and ω, and increasing c_2. This is especially interesting as the agent has learned to initially maintain stable CP values by itself, without the need to introduce such an inductive prior artificially. Such behaviour suggests more exploration during the first fifth of the search, while keeping the particles mostly within the feasible search space (Figure 9f). The effect is also reflected by the particle velocities in Figure 10f and swarm diversity (Figure 11f), both of which stay low initially.

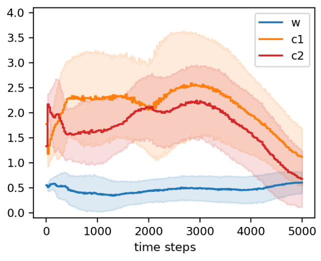
According to Table 4, the best performing sac_$n_t = 10$ improves upon the best performing baseline (baseline_tvac) by 33% for the training set. The test-set performance improvement is less significant at 13%. Self-adaptation using a SAC agent is, therefore, promising, because it improves on the baseline despite the velocity explosion.

Table 4. SAC-SAPSO normalised global best solution quality.

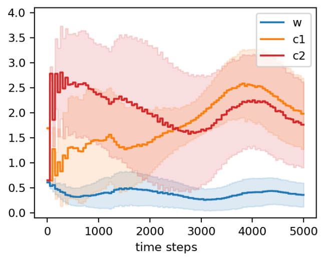
	Training		**Testing**	
	μ	σ	μ	σ
sac_n_t = **10**	0.0589	0.1338	0.2585	0.2602
sac_n_t = **125**	0.0628	0.1299	0.1811	0.192
sac_n_t = **25**	0.0687	0.1475	0.2415	0.1712
sac_n_t = **100**	0.0693	0.1599	0.2363	0.1905
sac_n_t = **auto**	0.0693	0.1529	0.2373	0.1675
sac_n_t = **50**	0.0737	0.1609	0.2342	0.1998



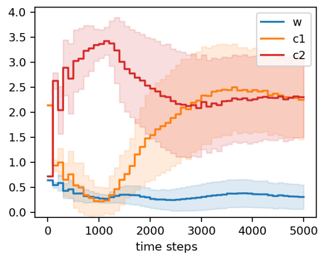
(**a**) $n_t = 10$



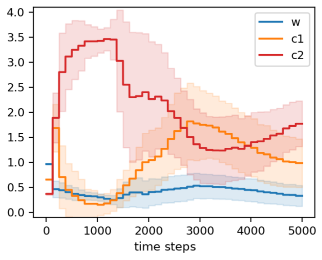
(**b**) $n_t = 25$



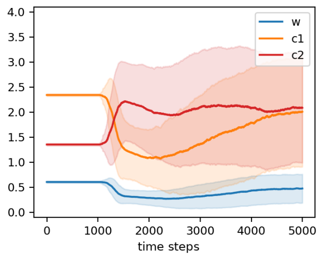
(**c**) $n_t = 50$



(**d**) $n_t = 100$

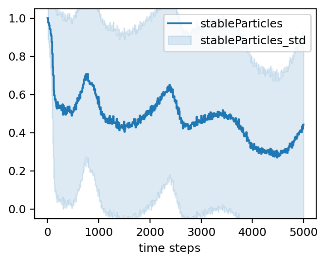


(**e**) $n_t = 125$

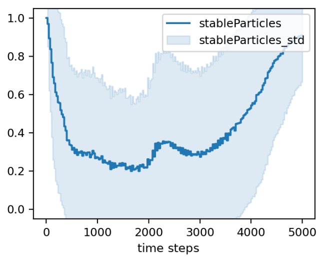


(**f**) n_t =auto

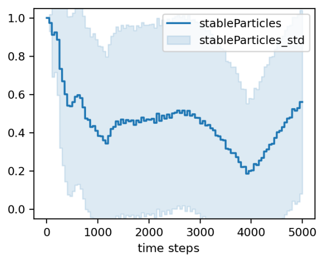
Figure 7. Control parameter values for SAC.



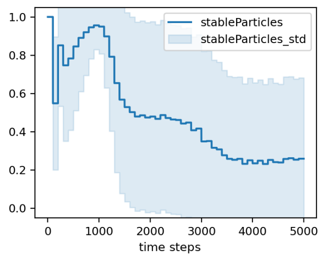
(**a**) $n_t = 10$



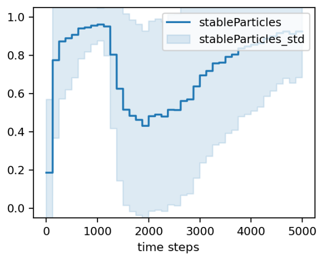
(**b**) $n_t = 25$



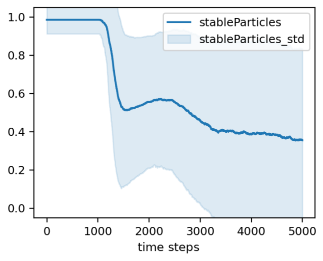
(**c**) $n_t = 50$



(**d**) $n_t = 100$

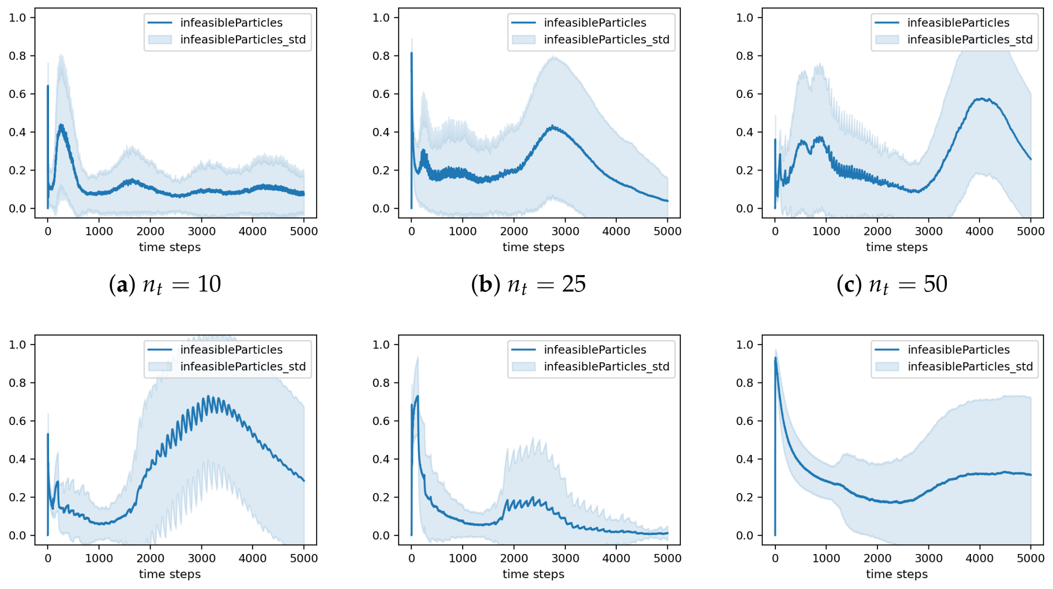


(**e**) $n_t = 125$



(**f**) n_t =auto

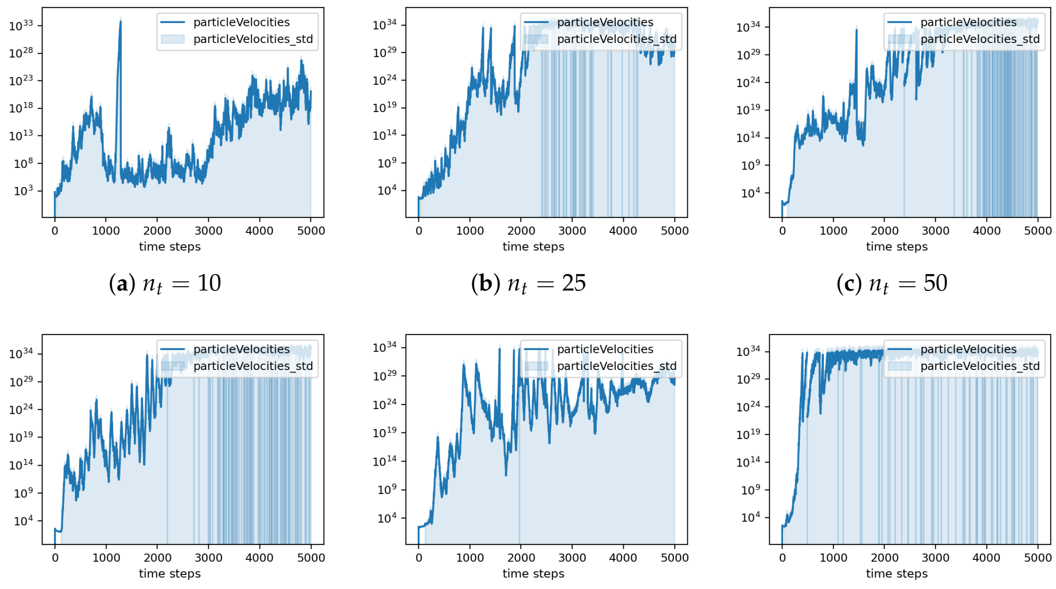
Figure 8. Stable particles for SAC.



(**a**) $n_t = 10$ (**b**) $n_t = 25$ (**c**) $n_t = 50$

(**d**) $n_t = 100$ (**e**) $n_t = 125$ (**f**) $n_t =$auto

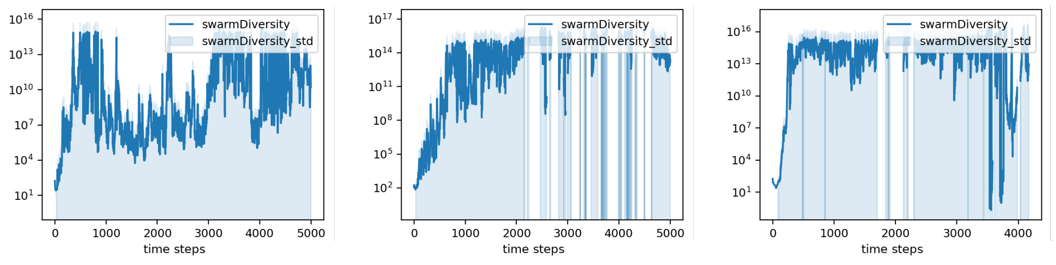
Figure 9. Infeasible particles for SAC.



(**a**) $n_t = 10$ (**b**) $n_t = 25$ (**c**) $n_t = 50$

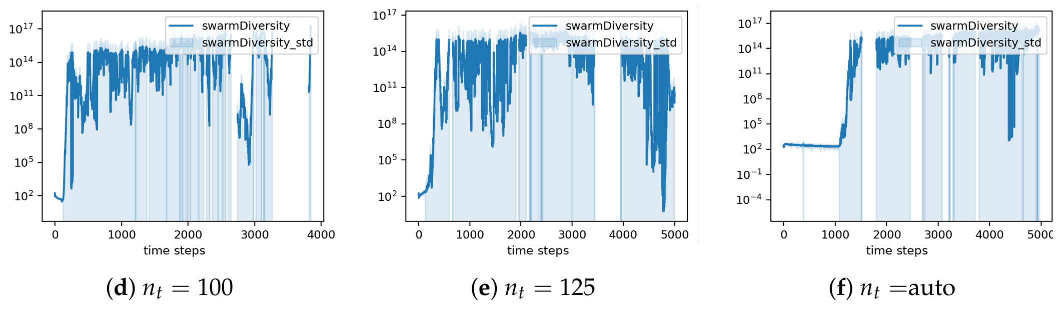
(**d**) $n_t = 100$ (**e**) $n_t = 125$ (**f**) $n_t =$auto

Figure 10. Particle velocities for SAC.



(**a**) $n_t = 10$ (**b**) $n_t = 25$ (**c**) $n_t = 50$

Figure 11. *Cont.*

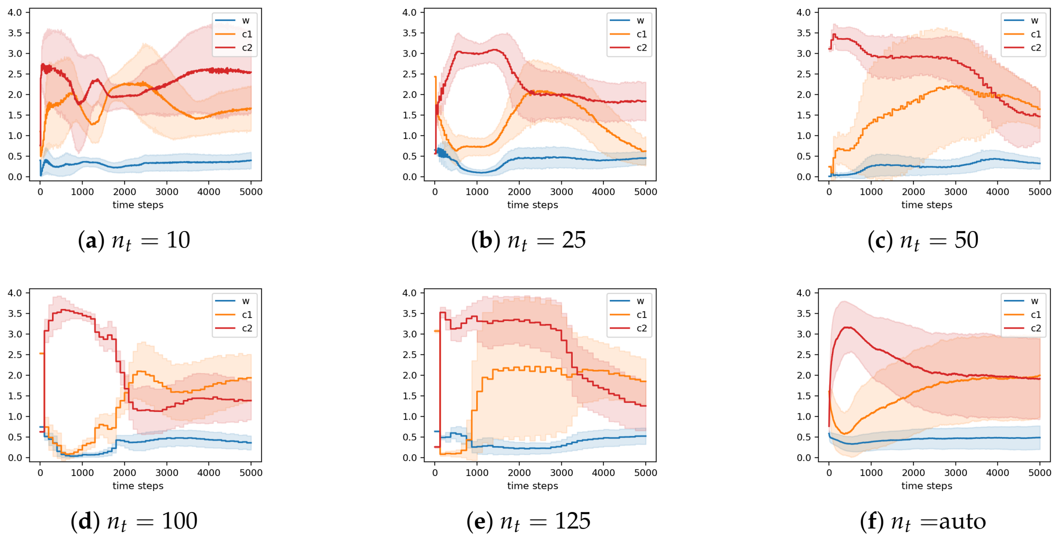


(**d**) $n_t = 100$ (**e**) $n_t = 125$ (**f**) n_t =auto

Figure 11. Swarm diversity (log) for SAC.

5.3. Soft Actor-Critic Control Parameter Adaptation with Velocity Clamping

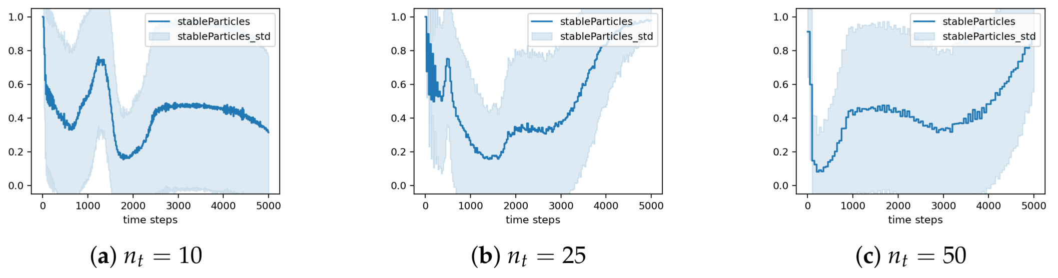
To counter the velocity explosion in Section 5.2, velocity clamping limits the maximum velocity of each particle, as discussed in Section 2.4. The velocity-clamped SAC-SAPSO results in the CP configurations are shown in Figure 12. Once again, multiple experiments are conducted for various values of n_t (vc_sac_$n_t = k$), as well as for the auto-tuned value of n_t (vc_sac_n_t =auto), with results given in Table 5 (refer to Figures 13–16). Figures 15 and 16 confirm that velocity clamping successfully constrains particle velocities and, therefore, also swarm diversity, as no explosion is present. Table 5 shows that velocity clamping has a massive effect on performance, which is to be expected because CP adjustments can have a more fine-grained effect on particles which reside within the boundaries of the search space than over those which overshoot the search space by many orders of magnitude. The best-performing vc_sac_$n_t = 125$ improves on the best-performing baseline (baseline_tvac) by 52% for the training set; on the test set, the improvement is 69%.



(**a**) $n_t = 10$ (**b**) $n_t = 25$ (**c**) $n_t = 50$

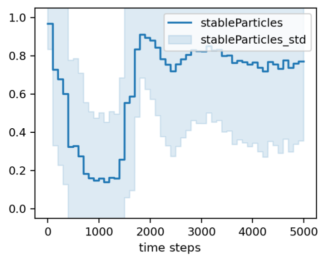
(**d**) $n_t = 100$ (**e**) $n_t = 125$ (**f**) n_t =auto

Figure 12. Control parameter values for SAC (velocity clamped).

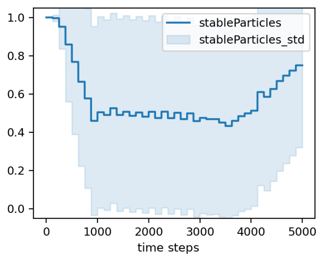


(**a**) $n_t = 10$ (**b**) $n_t = 25$ (**c**) $n_t = 50$

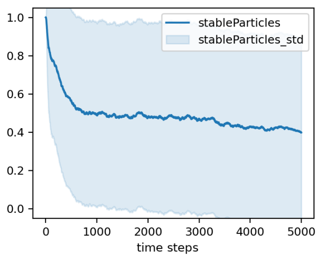
Figure 13. *Cont.*



(**d**) $n_t = 100$

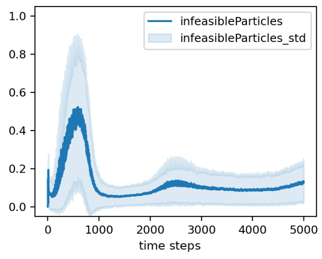


(**e**) $n_t = 125$

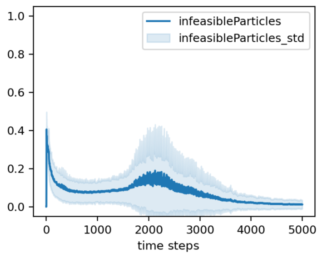


(**f**) n_t =auto

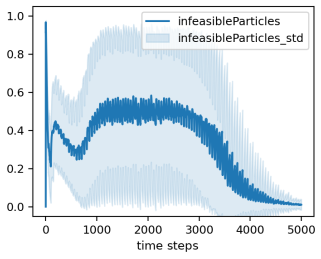
Figure 13. Stable particles for SAC (velocity clamped).



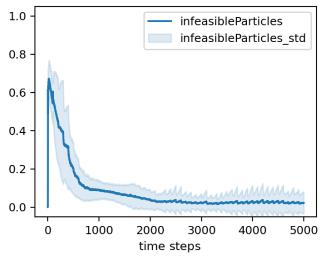
(**g**) $n_t = 10$



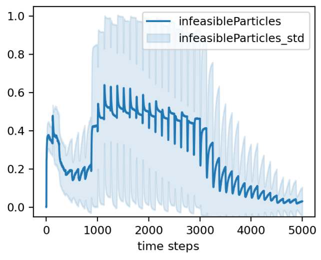
(**h**) $n_t = 25$



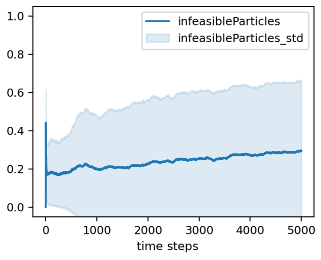
(**i**) $n_t = 50$



(**j**) $n_t = 100$

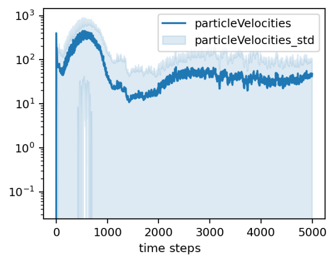


(**k**) $n_t = 125$

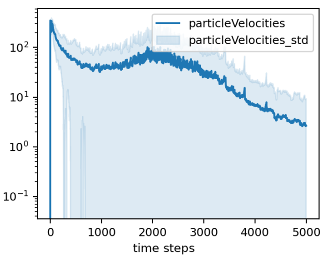


(**l**) n_t =auto

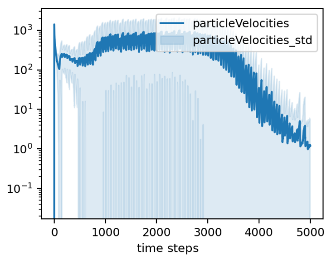
Figure 14. Infeasible particles for SAC (velocity clamped).



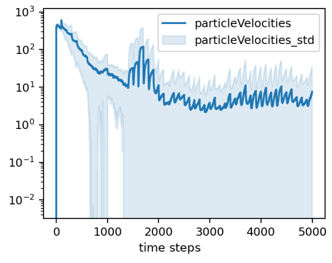
(**a**) $n_t = 10$



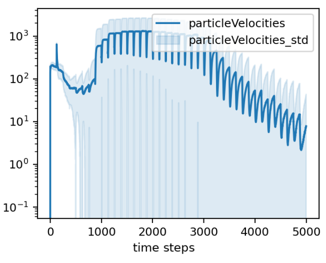
(**b**) $n_t = 25$



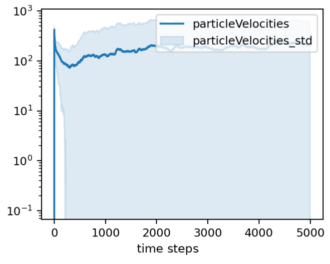
(**c**) $n_t = 50$



(**d**) $n_t = 100$

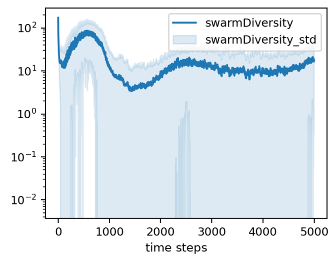


(**e**) $n_t = 125$

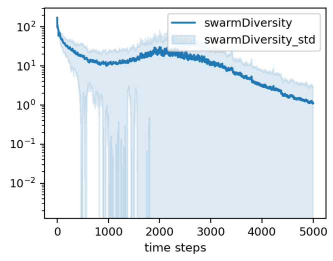


(**f**) n_t =auto

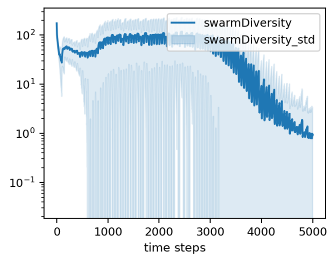
Figure 15. Particle velocities for SAC (velocity clamped).



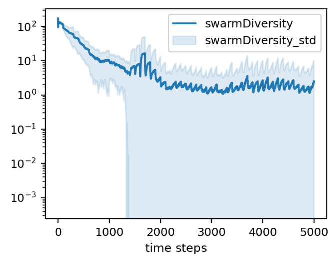
(**a**) $n_t = 10$



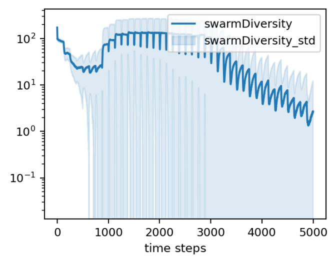
(**b**) $n_t = 25$



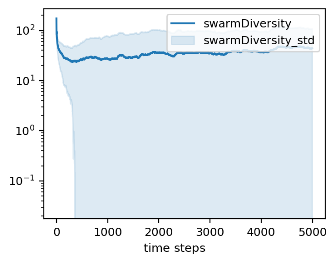
(**c**) $n_t = 50$



(**d**) $n_t = 100$



(**e**) $n_t = 125$



(**f**) n_t =auto

Figure 16. Swarm diversity (log) for SAC (velocity clamped).

Table 5. SAC-SAPSO normalised global best solution quality (velocity clamped).

	Training		**Testing**	
	μ	σ	μ	σ
vc_sac_n_t = **125**	0.0428	0.1115	0.0919	0.0892
vc_sac_n_t = **50**	0.0431	0.103	0.0689	0.0575
vc_sac_n_t = **100**	0.0473	0.1143	0.1089	0.0828
vc_sac_n_t = **auto**	0.049	0.109	0.228	0.1628
vc_sac_n_t = **10**	0.0554	0.1219	0.1931	0.1902
vc_sac_n_t = **25**	0.0559	0.1239	0.1443	0.1373

5.4. Overview of Results

Table 6 provides an overview of performance, with the normalised global best solution quality ranked from best to worst. The improvement made by the SAC-SAPSO above the baseline PSO is clear, as is the improvement which results from velocity clamping.

Preceding velocity clamping, the best performing sac_$n_t = 10$ demonstrates a 34% improvement above the constant baseline, and a 33% improvement above the time-variant baseline. On the test set, sac_$n_t = 10$ outperforms the constant baseline by 32% and the time-variant baseline by 21%. The best performing vc_sac_$n_t = 125$ improves on both the constant and time-variant baselines by 52% for the training set; on the test set, the improvement is 76% and 72%, respectively. Interestingly, vc_sac_$n_t = 50$ actually performs better than vc_sac_$n_t = 125$ on the test set, outperforming the constant baseline by 82% and the time-variant baseline by 79%.

Table 6. Comparison of normalised global best solution quality.

	Training		**Testing**	
	μ	σ	μ	σ
vc_sac_n_t = **125**	0.0428	0.1115	0.0919	0.0892
vc_sac_n_t = **50**	0.0431	0.103	0.0689	0.0575
vc_sac_n_t = **100**	0.0473	0.1143	0.1089	0.0828
vc_sac_n_t = **auto**	0.049	0.109	0.228	0.1628

Table 6. *Cont.*

	Training		**Testing**	
	μ	σ	μ	σ
vc_sac_n_t = 10	0.0554	0.1219	0.1931	0.1902
vc_sac_n_t = 25	0.0559	0.1239	0.1443	0.1373
sac_n_t = 10	0.0589	0.1338	0.2585	0.2602
sac_n_t = 125	0.0628	0.1299	0.1811	0.192
sac_n_t = 25	0.0687	0.1475	0.2415	0.1712
sac_n_t = 100	0.0693	0.1599	0.2363	0.1905
sac_n_t = auto	0.0693	0.1529	0.2373	0.1675
sac_n_t = 50	0.0737	0.1609	0.2342	0.1998
baseline_tvac	0.0883	0.1894	0.2926	0.2037
baseline_timevariant	0.0888	0.1991	0.3287	0.3052
baseline_constant	0.0893	0.1856	0.3802	0.2822
baseline_random	0.1044	0.2032	0.3812	0.3125

6. Conclusions

This paper presented a novel approach to the automatic adaptation of control parameters (CPs) in particle swarm optimisation (PSO). The approach is based on the soft-actor critic (SAC) algorithm, which learns a policy for the dynamic adaptation of CPs during the search process. Furthermore, velocity clamping prevents particle velocities from growing unboundedly. In conclusion, the velocity-clamped soft-actor critic self-adaptive particle swarm optimisation (SAC-SAPSO) algorithm successfully adjusted the CPs. The SAC-SAPSO achieved an improvement in performance of 52% relative to the constant CP baseline (76% on the training set), and 52% relative to the time-variant baseline (72% on the training set). The value of n_t can, of course, still be considered as a CP, implying a reduction in the total number of parameters from three to one, in addition to the improvement in performance. If a search process without CPs is desired, the velocity-clamped SAC-SAPSO which automatically adapts n_t (vc_sac_n_t = auto) can be used. Velocity clamping proved useful, firstly to improve performance, but also to prevent the search process from becoming unstable with particle velocities growing unboundedly. All methods displayed a degree of invariance with respect to the choice of n_t, because all SAC methods outperformed all baselines, and all velocity-clamped SAC methods outperform all SAC methods.

Finally, the environment and agent were configured in an agnostic fashion; many swarm intelligence or evolutionary algorithms with tunable parameters can, therefore, be "plugged into" the framework in order to find an adaptive policy for its parameters.

7. Future Work

The environment currently only generates particle velocities, percentage stable particles, percentage feasible particles, and percentage completion of the search for the agent to observe. The agent could theoretically reconstruct a much more nuanced representation of the fitness landscape if it received positions or objective function values also, thereby leading to a more landscape-aware PSO. However, to counteract an explosion in computational complexity, dimensionality reduction techniques such as principal component analysis (PCA) or linear discriminant analysis (LDA) might have to be used to obtain a lower-dimensional representation of particle positions.

Furthermore, the action space $\mathcal{A}$ can be extended so that the agent may adjust the CP configuration per particle.

The algorithm was evaluated for only 5000 time-steps. An alternative option is to not set a hard limit on the maximum time-steps, thereby allowing the agent to use as few or as many time-steps as it deems necessary for the problem at hand. This may require the introduction of negative reward if the agent begins to exceed the computational budget. Similarly, the search could be terminated if the swarm converges or if improvement has stalled. The aforementioned could thus introduce adaptive behaviour to the search length also.

Similarly, negative reward can be used to punish the agent proportionally to the number of particles which leave the feasible search space. While this will definitely lead to more stable CP configurations and less infeasible particles, care must be taken when introducing such an inductive bias, as it might also degrade performance since the agent would then no longer optimise purely with respect to performance. However, artificially encouraging convergent search behaviour might very well lead the agent to find better solutions, as it helps to prevent the SAC from choosing a CP configuration which lies outside of the convergent CP region.

The performance of the SAC-SAPSO on dynamic optimisation problems will also be studied.

Author Contributions: Conceptualisation, D.v.E. and A.E.; methodology, D.v.E. and A.E.; software, D.v.E.; validation, D.v.E.; formal analysis, D.v.E.; investigation, D.v.E. and A.E.; resources, D.v.E.; data curation, D.v.E.; writing—original draft preparation, D.v.E.; writing—review and editing, A.E.; visualisation, D.v.E.; supervision, A.E.; project administration, D.v.E. and A.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The datasets generated during and/or analysed during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CP	Control Parameter
PSO	Particle Swarm Optimisation

Appendix A. Benchmark Functions

In the process of designing and validating algorithms for swarm intelligence, it is crucial to employ a diverse and challenging set of optimisation functions. This appendix details the optimisation function set used in the empirical evaluation of the proposed RL-based SAPSO algorithm. These functions are characteristically diverse, encompassing continuous, non-continuous, differentiable, non-differentiable, separable, non-separable, unimodal, and multimodal functions. This wide-ranging collection of functions ensures a robust evaluation of the algorithm under a spectrum of optimisation scenarios, providing a comprehensive understanding of its performance and adaptability.

The optimisation function equations are given below. Their characteristics follow in Table A1, with the test set functions at the end in **bold** (https://infinity77.net/global_optimization/, accessed on 1 November 2023).

- Ackley 1 for $x_j \in [-32, 32]$:

$$f(\mathbf{x}) = -20e^{-0.2\sqrt{\frac{1}{n_x}\sum_{j=1}^{n_x} x_j^2}} - e^{\frac{1}{n_x}\sum_{j=1}^{n_x}\cos(2\pi x_j)} + 20 + e \tag{A1}$$

- Alpine 1 for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} |x_j \sin(x_j) + 0.1x_j| \tag{A2}$$

- Bohachevsky 1 for $x_j \in [-15, 15]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1} (x_j^2 + 2x_{j+1}^2 - 0.3\cos(3\pi x_j) - 0.4\cos(4\pi x_{j+1}) + 0.7) \tag{A3}$$

- Bonyadi Michalewicz for $x_j \in [-5, 5]$:

$$f(\mathbf{x}) = \frac{\prod_{j=1}^{n_x}(x_j + 1)}{\prod_{j=1}^{n_x}((x_j + 1)^2 + 1)} \tag{A4}$$

- Brown for $x_j \in [-1, 1]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1} (x_j^2)^{(x_{j+1}^2+1)} + (x_{j+1}^2)^{(x_j^2+1)} \tag{A5}$$

- Cosine Mixture for $x_j \in [-1, 1]$:

$$f(\mathbf{x}) = 0.1\sum_{j=1}^{n_x} \cos(5\pi x_j) + \sum_{j=1}^{n_x} x_j^2 \tag{A6}$$

- Cross Leg Table for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = -\frac{1}{\left(\left|e^{|100-\frac{\sqrt{\sum_{j=1}^{n_x} x_j^2}}{\pi}|}\prod_{j=1}^{n_x}\sin(x_j)\right| + 1\right)^{0.1}} \tag{A7}$$

- Deflected Corrugated Spring for $x_j \in [0, 2\alpha]$, $K = 5$, $\alpha = 5$:

$$f(\mathbf{x}) = 0.1\sum_{j=1}^{n_x}\left[(x_j - \alpha)^2 - \cos\left(K\sqrt{\sum_{j=1}^{n_x}(x_j - \alpha)^2}\right)\right] \tag{A8}$$

- Discuss for $x_j \in [-100, 100]$:

$$f(\mathbf{x}) = 10^6 x_1^2 + \sum_{i=2}^{n_x} x_j^2 \tag{A9}$$

- Drop Wave for $x_j \in [-5.12, 5.12]$:

$$f(\mathbf{x}) = -\frac{1 + \cos(12\sqrt{\sum_{j=1}^{n_x} x_j^2})}{2 + 0.5\sum_{j=1}^{n_x} x_j^2} \tag{A10}$$

- Egg Crate for $x_j \in [-5, 5]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} x_j^2 + 24\sum_{j=1}^{n_x} \sin(x_j)^2 \tag{A11}$$

- Egg Holder for $x_j \in [-512, 512]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1}\Big(-(x_{j+1} + 47)\sin(\sqrt{|x_{j+1} + x_j/2 + 47|}) - x_j\sin(\sqrt{|x_j - (x_{j+1} + 47)|})\Big) \tag{A12}$$

- Elliptic for $x_j \in [-100, 100]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} (10^6)^{\frac{j-1}{n_x-1}} x_j^2 \tag{A13}$$

- Exponential for $x_j \in [-1, 1]$:

$$f(\mathbf{x}) = -e^{-0.5 \sum_{j=1}^{n_x} x_j^2} \tag{A14}$$

- Giunta for $x_j \in [-1, 1]$:

$$f(\mathbf{x}) = 0.6 + \sum_{j=1}^{2} \left(\sin(\frac{16}{15} x_j - 1) + \sin(\frac{16}{15} x_j - 1)^2 + \frac{1}{50} \sin(4(\frac{16}{15} x_j - 1)) \right) \tag{A15}$$

- Holder Table 1 for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = -\left| \left(\prod_{j=1}^{n_x} \cos(x_j) \right) e^{|1 - \sqrt{\sum_{j=1}^{n_x} x_j} / \pi|} \right| \tag{A16}$$

- Lanczos 3 for $x_j \in [-20, 20]$:

$$f(\mathbf{x}) = \prod_{j=1}^{n_x} \text{sinc}(x_j) \text{sinc}(x_j / k) \tag{A17}$$

where $\text{sinc}(x) = \frac{sin(x)}{x}$

- Levy 3 for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = \sin^2(\pi y_1) + \sum_{j=1}^{n_x - 1} (y_j - 1)^2 [1 + 10 \sin^2(\pi y_{j+1})] + (y_{n_x} - 1)^2 \tag{A18}$$

where

$$y_j = 1 + \frac{x_j - 1}{4}$$

- Levy-Montalvo 2 for $x_j \in [-5, 5]$:

$$f(\mathbf{x}) = 0.1 \sin(3\pi x_1)^2 + 0.1 \sum_{j=1}^{n_x - 1} (x_j - 1)^2 \left[\sin^2(3\pi x_{j+1}) + 1 \right] + 0.1 (x_{n_x} - 1)^2 \left[\sin^2(2\pi x_{n_x}) + 1 \right] \tag{A19}$$

- Michalewicz for $x_j \in [0, \pi]$:

$$f(\mathbf{x}) = -\sum_{j=1}^{n_x} \sin(x_j) \left(\sin\left(\frac{j x_j^2}{\pi} \right) \right)^{2m} \tag{A20}$$

where $m = 10$.

- Mishra 1 for $x_j \in [0, 1]$:

$$f(\mathbf{x}) = \left(1 + n_x - \sum_{j=1}^{n_x - 1} x_j \right)^{(n_x - \sum_{j=1}^{n_x - 1} x_j)} \tag{A21}$$

- Mishra 4 for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = \sqrt{\left|\sin\sqrt{|\sum_{j=1}^{n_x} x_j^2|}\right|} + 0.01\sum_{j=1}^{n_x} x_j \tag{A22}$$

- Needle Eye for $x_j \in [-10, 10]$ and $eye = 0.0001$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } |x_j| < eye \ \ \forall j \\ \sum_{j=1}^{n_x}(100 + |x_j|) & \text{if } |x_j| > eye \\ 0 & \text{otherwise} \end{cases} \tag{A23}$$

- Norwegian for $x_j \in [-1.1, 1.1]$:

$$f(\mathbf{x}) = \prod_{j=1}^{n_x}\left(\cos(\pi x_j^3)\left(\frac{99 + x_j}{100}\right)\right) \tag{A24}$$

- Pathological for $x_j \in [-100, 100]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1}\left(\frac{\sin^2\sqrt{100x_j^2 + x_{j+1}^2} - 0.5}{0.5 + 0.001(x_j - x_{j+1})^4}\right) \tag{A25}$$

- Penalty 1 for $x_j \in [-50, 50]$:

$$\begin{aligned} f(\mathbf{x}) &= \frac{\pi}{30}\left[10\sin^2(\pi y_1) + \sum_{j=1}^{n_x-1}(y_j - 1)^2\left[1 + 10\sin^2(\pi y_{j+1})\right] + (y_{n_x} - 1)^2\right] \\ &+ \sum_{j=1}^{n_x} u(x_j, 10, 100, 4) \end{aligned} \tag{A26}$$

where

$$y_j = 1 + \frac{1}{4}(x_j + 1)$$

and

$$u(x_j, a, k, m) = \begin{cases} k(x_j - a)^m & \text{if } \ x_j > a \\ 0 & \text{if } \ -a \le x_j \le \ a \\ k(-x_j - a)^m & \text{if } \ x_j < -a \end{cases}$$

- Penalty 2 for $x_j \in [-50, 50]$:

$$\begin{aligned} f(\mathbf{x}) &= 0.1\Bigg(\sin^2(3\pi x_1) + \sum_{j=1}^{n_x-1}(x_j - 1)^2\left[1 + \sin^2(3\pi x_{j+1})\right] \\ &+ (x_{n_x} - 1)^2\left[1 + \sin^2(2\pi x_{n_x})\right]\Bigg) + \sum_{j=1}^{n_x} u(x_j, 5, 100, 4) \end{aligned} \tag{A27}$$

where

$$u(x_j, a, k, m) = \begin{cases} k(x_j - a)^m & \text{if } \ x_j > a \\ 0 & \text{if } \ -a \le x_j \le \ a \\ k(-x_j - a)^m & \text{if } \ x_j < -a \end{cases}$$

- Periodic for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = 1 + \sum_{j=1}^{n_x} \sin^2(x_j) - 0.1e^{-\sum_{j=1}^{n_x} x_j^2} \tag{A28}$$

- Pinter for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} jx_j^2 + \sum_{j=1}^{n_x} 20j \sin^2 A + \sum_{j=1}^{n_x} j \log_{10}(1 + jB^2) \tag{A29}$$

where

$$A = x_{j-1} \sin x_j + \sin x_{j+1}$$

$$B = x_{j-1}^2 - 2x_j + 3x_{j+1} - \cos x_j + 1$$

- Price 2 for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = 1 + \sum_{j=1}^{n_x} \sin^2(x_j) - 0.1e^{-\sum_{j=1}^{n_x} x_j^2} \tag{A30}$$

- Qings for $x_j \in [-500, 500]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} (x_j^2 - j)^2 \tag{A31}$$

- Quadric for $x_j \in [-100, 100]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} \left(\sum_{l=1}^{j} x_l \right)^2 \tag{A32}$$

- Quintic for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} \left| x_j^5 - 3x_j^4 + 4x_j^3 + 2x_j^2 - 10x_j - 4 \right| \tag{A33}$$

- Rana for $x_j \in [-500, 500]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x - 1} (x_{j+1} + 1) \cos(t_2) \sin(t_1) + x_j \cos(t_1) \sin(t_2) \tag{A34}$$

where

$$t_1 = \sqrt{|x_{j+1} + x_j + 1|} \quad \text{and} \quad t_2 = \sqrt{|x_{j+1} - x_j + 1|}$$

- Rastrigin for $x_j \in [-5.12, 5.12]$

$$f(\mathbf{x}) = 10n + \sum_{j=1}^{n_x} \left(x_j^2 - 10 \cos(2\pi x_j) \right) \tag{A35}$$

- Ripple 25 for $x_j \in [0, 1]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} -e^{-2 \log_2 (\frac{x_j - 0.1}{0.8})^2 \left(\sin^6(5\pi x_j)) \right)} \tag{A36}$$

- Rosenbrock for $x_j \in [-30, 30]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1} \left(100(x_{j+1} - x_j^2)^2 + (x_j - 1)^2\right) \tag{A37}$$

- Salomon for $x_j \in [-100, 100]$:

$$f(\mathbf{x}) = -\cos(2\pi \sum_{j=1}^{n_x} x_j^2) + 0.1\sqrt{\sum_{j=1}^{n_x} x_j^2} + 1 \tag{A38}$$

- Schaffer 4 for $x_j \in [-100, 100]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1} \left(0.5 + \frac{\cos^2(\sin(x_j^2 - x_{j+1}^2))^2 - 0.5}{(1 + 0.001(x_j^2 + x_{j+1}^2))^2}\right) \tag{A39}$$

- Schubert 4 for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} \sum_{j=1}^{5} j\cos((j+1)x_j + j) \tag{A40}$$

- Schwefel 1 for $x_j \in [-100, 100]$ and $\alpha = \sqrt{\pi}$:

$$f(\mathbf{x}) = \left(\sum_{j=1}^{n_x} x_j^2\right)^{\alpha} \tag{A41}$$

- Sine Envelope for $x_j \in [-100, 100]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1} \left(0.5 + \frac{\sin^2\sqrt{x_j^2 + x_{j+1}^2} - 0.5}{(1 + 0.001(x_j^2 + x_{j+1}^2))^2}\right) \tag{A42}$$

- Sinusoidal for $x_j \in [0, 180]$:

$$f(\mathbf{x}) = -\left[A\prod_{j=1}^{n_x} \sin(x_j - z) + \prod_{j=1}^{n_x} \sin(B(x_j - z))\right] \tag{A43}$$

- Step Function 3 for $x_j \in [-5.12, 5.12]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} \lfloor x_j^2 \rfloor \tag{A44}$$

- Stretched V Sine Wave for $x_j \in [-10, 10]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x-1} (x_j^2 + x_{j+1}^2)^{0.25}[\sin^2(50(x_j^2 + x_{j+1}^2)^{0.1}) + 0.1] \tag{A45}$$

- Trid for $x_j \in [-20, 20]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} (x_j - 1)^2 - \sum_{i=2}^{n_x} x_j x_{j-1} \tag{A46}$$

- Trigonometric for $x_j \in [0, \pi]$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} \left[n - \sum_{j=1}^{n_x} \cos(x_j) + i(1 - \cos(x_j) - sin(x_j)) \right]^2 \tag{A47}$$

- Wavy for $x_j \in [-\pi, \pi]$:

$$f(\mathbf{x}) = 1 - \frac{1}{n_x} \sum_{j=1}^{n_x} \cos(kx_j) e^{\frac{-x_j^2}{2}} \tag{A48}$$

- Weierstrass for $j_{max} = 20, a = 0.5, b = 3$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} \left(\sum_{j=0}^{j_{max}} a^j \cos(2\pi b^j (x_j + 0.5)) - n \sum_{j=1}^{j_{max}} a^j \cos(\pi b^j) \right) \tag{A49}$$

- Vincent for $x_j \in [0.25, 10]$:

$$f(\mathbf{x}) = -\sum_{j=1}^{n_x} \sin(10 \log(x)) \tag{A50}$$

- Xin-She Yang 1 for $\epsilon_i \sim U(0, 1)$:

$$f(\mathbf{x}) = \sum_{j=1}^{n_x} \epsilon_i |x_j|^i \tag{A51}$$

- Xin-She Yang 2 for $x_j \in [-2\pi, 2\pi]$:

$$f(\mathbf{x}) = \left(\sum_{j=1}^{n_x} |x_j| \right) e^{-\sum_{j=1}^{n_x} \sin(x_j^2)} \tag{A52}$$

Table A1. Function characteristics (C = continuous, NC = non-continuous, D = differentiable, ND = non-differentiable, S = separable, NS = non-separable, MM = multi-modal, UM = unimodal).

Function	Equation	Cont.	Diff.	Sep.	Mod.
Ackley 1	(A1)	C	D	NS	MM
Alpine 1	(A2)	C	ND	S	MM
Bohachevsky1	(A3)	C	D	S	MM
Bonyadi-Michalewicz	(A4)	C	D	NS	MM
Brown	(A5)	C	D	NS	UM
Cosine Mixture	(A6)	C	D	S	MM
Deflected Corrugated Spring	(A8)	C	D	S	MM
Discuss	(A9)	C	D	S	UM
Drop Wave	(A10)	C	D	NS	MM
Egg crate	(A11)	C	D	S	MM
Egg Holder	(A12)	C	ND	NS	MM
Elliptic	(A13)	C	D	S	UM
Exponential	(A14)	C	D	NS	UM
Giunta	(A15)	C	D	S	MM
Holder Table 1	(A16)	C	ND	NS	MM
Levy 3	(A18)	C	D	NS	MM
Levy–Montalvo 2	(A19)	C	D	NS	MM
Mishra 1	(A21)	C	D	NS	MM
Mishra 4	(A22)	C	ND	NS	MM
Needle Eye	(A23)	C	ND	S	MM

Table A1. *Cont.*

Function	Equation	Cont.	Diff.	Sep.	Mod.
Norwegian	(A24)	C	D	NS	MM
Pathological	(A25)	C	D	NS	MM
Penalty 1	(A26)	C	D	NS	MM
Penalty 2	(A27)	C	D	NS	MM
Periodic	(A28)	C	D	S	MM
Pinter 2	(A29)	C	D	NS	MM
Price 2	(A30)	C	D	NS	MM
Qings	(A31)	C	D	S	MM
Quadric	(A32)	C	D	NS	UM
Quintic	(A33)	C	ND	S	MM
Rana	(A34)	C	ND	NS	MM
Rastrigin	(A35)	C	D	S	MM
Ripple 25	(A36)	C	D	S	MM
Rosenbrock	(A37)	C	D	NS	MM
Salomon	(A38)	C	D	NS	MM
Schubert 4	(A40)	C	D	S	MM
Schwefel 1	(A41)	C	D	S	UM
Sinusoidal	(A43)	C	D	NS	MM
Step Function 3	(A44)	NC	ND	S	MM
Trid	(A46)	C	D	NS	MM
Trigonometric	(A47)	C	D	NS	MM
Vincent	(A50)	C	D	S	MM
Weierstrass	(A49)	C	D	S	MM
Xin-She Yang 1	(A51)	NC	ND	S	MM
Xin-She Yang 2	(A52)	C	ND	NS	MM
Cross Leg Table	(A7)	NC	ND	NS	MM
Lanczos 3	(A17)	C	D	NS	MM
Michalewicz	(A20)	C	D	S	MM
Schaffer 4	(A39)	C	D	NS	MM
Sine Envelope	(A42)	C	D	NS	MM
Stretched V Sine Wave	(A45)	C	D	NS	MM
Wavy	(A48)	C	D	S	UM

References

1. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
2. Beielstein, T.; Parsopoulos, K.E.; Vrahatis, M.N. *Tuning PSO Parameters Through Sensitivity Analysis*; Technical Report Interner Bericht des Sonderforschungsbereichs (SFB) 531 Computational Intelligence No.CI-124/02; Universitätsbibliothek Dortmund: Dortmund, Germany, 2002.
3. Van den Bergh, F.; Engelbrecht, A.P. A study of particle swarm optimization particle trajectories. *Inf. Sci.* **2006**, *176*, 937–971. [CrossRef]
4. Bonyadi, M.R.; Michalewicz, Z. Impacts of coefficients on movement patterns in the particle swarm optimization algorithm. *IEEE Trans. Evol. Comput.* **2016**, *21*, 378–390. [CrossRef]
5. Bratton, D.; Kennedy, J. Defining a standard for particle swarm optimization. In Proceedings of the IEEE Swarm Intelligence Symposium, Honolulu, HI, USA, 1–5 April 2007; pp. 120–127.
6. Jiang, M.; Luo, Y.; Yang, S. Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. *Inf. Process. Lett.* **2007**, *102*, 8–16. [CrossRef]
7. Harrison, K.; Engelbrecht, A.P.; Ombuki-Berman, B. Self-adaptive particle swarm optimization: A review and analysis of convergence. *Swarm Intell.* **2018**, *12*, 187–226. [CrossRef]
8. Harrison, K.; Engelbrecht, A.; Ombuki-Berman, B. Inertia Control Strategies for Particle Swarm Optimization: Too Much Momentum, Not Enough Analysis. *Swarm Intell.* **2016**, *10*, 267–305. [CrossRef]
9. Engelbrecht, A.P. Roaming Behavior of Unconstrained Particles. In Proceedings of the BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence, Ipojuca, Brazil, 8–11 September 2013; pp. 104–111.
10. Harrison, K.R.; Engelbrecht, A.P.; Ombuki-Berman, B.M. Optimal parameter regions and the time-dependence of control parameter values for the particle swarm optimization algorithm. *Swarm Evol. Comput.* **2018**, *41*, 20–35. [CrossRef]
11. Ratnaweera, A.; Halgamuge, S.; Watson, H. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Trans. Evol. Comput.* **2004**, *8*, 240–255. [CrossRef]

12. Leonard, B.J.; Engelbrecht, A.P. On the optimality of particle swarm parameters in dynamic environments. In Proceedings of the IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 1564–1569.
13. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
14. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.
15. Barsce, J.C.; Palombarini, J.A.; Martínez, E.C. Automatic tuning of hyper-parameters of reinforcement learning algorithms using Bayesian optimization with behavioral cloning. *arXiv* **2021**, arXiv:2112.08094.
16. Talaat, F.M.; Gamel, S.A. RL based hyper-parameters optimization algorithm (ROA) for convolutional neural network. *J. Ambient Intell. Humaniz. Comput.* **2022**, *13*, 3389–3402. [CrossRef]
17. Liu, X.; Wu, J.; Chen, S. Efficient hyperparameters optimization through model-based reinforcement learning with experience exploiting and meta-learning. *Soft Comput.* **2023**, *27*, 7051–7066. [CrossRef]
18. Wauters, T.; Verbeeck, K.; De Causmaecker, P.; Vanden Berghe, G. Boosting Metaheuristic Search Using Reinforcement Learning. In *Hybrid Metaheuristics*; Talbi, E.G., Ed.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 433–452.
19. Shi, Y.; Eberhart, R. A modified particle swarm optimizer. In Proceedings of the IEEE International Conference on Evolutionary Computation, Anchorage, AK, USA, 4–9 May 1998; Volume 6, pp. 69–73.
20. Clerc, M.; Kennedy, J. The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE Trans. Evol. Comput.* **2002**, *6*, 58–73. [CrossRef]
21. Sermpinis, G.; Theofilatos, K.; Karathanasopoulos, A.; Georgopoulos, E.F.; Dunis, C. Forecasting foreign exchange rates with adaptive neural networks using radial-basis functions and Particle Swarm Optimization. *Eur. J. Oper. Res.* **2013**, *225*, 528–540. [CrossRef]
22. Poli, R. Mean and Variance of the Sampling Distribution of Particle Swarm Optimizers During Stagnation. *IEEE Trans. Evol. Comput.* **2009**, *13*, 712–721. [CrossRef]
23. Poli, R.; Broomhead, D. Exact Analysis of the Sampling Distribution for the Canonical Particle Swarm Optimiser and Its Convergence during Stagnation. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, London, UK, 7–11 July 2007; Association for Computing Machinery: New York, NY, USA, 2007; pp. 134–141.
24. von Eschwege, D.; Engelbrecht, A. A Cautionary Note on Poli's Stability Condition for Particle Swarm Optimization. In Proceedings of the IEEE Swarm Intelligence Symposium, Mexico City, Mexico, 5–8 December 2023.
25. Oldewage, E.T.; Engelbrecht, A.P.; Cleghorn, C.W. The merits of velocity clamping particle swarm optimisation in high dimensional spaces. In Proceedings of the IEEE Symposium Series on Computational Intelligence, Honolulu, HI, USA, 27 November–1 December 2017; pp. 1–8.
26. Li, X.; Fu, H.; Zhang, C. A Self-Adaptive Particle Swarm Optimization Algorithm. In Proceedings of the International Conference on Computer Science and Software Engineering, Wuhan, China, 12–14 December 2008; Volume 5, pp. 186–189.
27. Dong, C.; Wang, G.; Chen, Z.; Yu, Z. A Method of Self-Adaptive Inertia Weight for PSO. In Proceedings of the International Conference on Computer Science and Software Engineering, Wuhan, China, 12–14 December 2008; Volume 1, pp. 1195–1198.
28. Xu, G. An Adaptive Parameter Tuning of Particle Swarm Optimization Algorithm. *Appl. Math. Comput.* **2013**, *219*, 4560–4569. [CrossRef]
29. Hashemi, A.; Meybodi, M. A note on the learning automata based algorithms for adaptive parameter selection in PSO. *Appl. Soft Comput.* **2011**, *11*, 689–705. [CrossRef]
30. Haarnoja, T.; Tang, H.; Abbeel, P.; Levine, S. Reinforcement Learning with Deep Energy-Based Policies. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 1352–1361.
31. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft Actor-Critic Algorithms and Applications. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.
32. Ziebart, B.D.; Maas, A.; Bagnell, J.A.; Dey, A.K. Maximum Entropy Inverse Reinforcement Learning. In Proceedings of the 23rd National Conference on Artificial Intelligence, Washington, DC, USA, 2008; Volume 3, pp. 1433–1438.
33. Maei, H.R.; Szepesvári, C.; Bhatnagar, S.; Precup, D.; Silver, D.; Sutton, R.S. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In Proceedings of the 22nd International Conference on Neural Information Processing Systems; Curran Associates Inc.: Red Hook, NY, USA, 2009; pp. 1204–1212.
34. Kullback, S. On information and sufficiency. *Ann. Math. Stat.* **1951**, *22*, 79–86. [CrossRef]
35. Van der Stockt, S.A.; Engelbrecht, A.P. Analysis of selection hyper-heuristics for population-based meta-heuristics in real-valued dynamic optimization. *Swarm Evol. Comput.* **2018**, *43*, 127–146. [CrossRef]
36. Grobler, J.; Engelbrecht, A.P.; Kendall, G.; Yadavalli, V.S.S. Alternative hyper-heuristic strategies for multi-method global optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, Barcelona, Spain, 18–23 July 2010; pp. 1–8.
37. Grobler, J.; Engelbrecht, A.P.; Kendall, G.; Yadavalli, V. Multi-method algorithms: Investigating the entity-to-algorithm allocation problem. In Proceedings of the IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 570–577.
38. Grobler, J.; Engelbrecht, A.P.; Kendall, G.; Yadavalli, V. Heuristic space diversity control for improved meta-hyper-heuristic performance. *Inf. Sci.* **2015**, *300*, 49–62. [CrossRef]

39. Nareyek, A. Choosing Search Heuristics by Non-Stationary Reinforcement Learning. In *Metaheuristics: Computer Decision-Making*; Springer: Boston, MA, USA, 2004; pp. 523–544.
40. Burke, E.K.; Kendall, G.; Soubeiga, E. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *J. Heuristics* **2003**, *9*, 451–470. [CrossRef]
41. Wirth, C.; Fürnkranz, J. EPMC: Every Visit Preference Monte Carlo for Reinforcement Learning. In Proceedings of the 5th Asian Conference on Machine Learning, Canberra, Australia, 13–15 November 2013; Volume 29, pp. 483–497.
42. Rummery, G.A.; Niranjan, M. *On-Line Q-Learning Using Connectionist Systems*; Technical Report CUED/F-INFENG/TR 166; Department of Engineering, University of Cambridge: Cambridge, UK, 1994.
43. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
44. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
45. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Harley, T.; Lillicrap, T.P.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML'16, New York, NY, USA, 20–22 June 2016; Volume 48, pp. 1928–1937.
46. Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.I.; Abbeel, P. Trust Region Policy Optimization. In Proceedings of the 31st International Conference on Machine Learning, Lille, France, 11 July 2015.
47. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
48. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. In Proceedings of the 4th International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
49. Fujimoto, S.; van Hoof, H.; Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Volume 80, pp. 1582–1591.
50. Olorunda, O.; Engelbrecht, A.P. Measuring exploration/exploitation in particle swarms using swarm diversity. In Proceedings of the IEEE Congress on Evolutionary Computation, Hong Kong, China, 1–6 June 2008; pp. 1128–1134.
51. Cleghorn, C.W.; Engelbrecht, A. Particle swarm optimizer: The impact of unstable particles on performance. In Proceedings of the IEEE Swarm Intelligence Symposium, Athens, Greece, 6–9 December 2016; pp. 1–7.
52. Goodfellow, I.J.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
53. Engelbrecht, A. Stability-Guided Particle Swarm Optimization. In *Swarm Intelligence (ANTS)*; Springer: Cham, Switzerland, 2022.
54. Harrison, K.R.; Engelbrecht, A.P.; Ombuki-Berman, B.M. An adaptive particle swarm optimization algorithm based on optimal parameter regions. In Proceedings of the IEEE Swarm Intelligence Symposium, Honolulu, HI, USA, 27 November–1 December 2017; pp. 1–8.