


Article

Deep Q-Network-Enhanced Self-Tuning Control of Particle Swarm Optimization

Oussama Aoun 

Laboratory of Advanced Research in Industrial and Logistic Engineering, Superior National School of Electricity and Mechanic (ENSEM), Hassan II University, Casablanca 20000, Morocco; o.aoun@ensem.ac.ma

Abstract: Particle Swarm Optimization (PSO) is a widespread evolutionary technique that has successfully solved diverse optimization problems across various application fields. However, when dealing with more complex optimization problems, PSO can suffer from premature convergence and may become stuck in local optima. The primary goal is accelerating convergence and preventing solutions from falling into these local optima. This paper introduces a new approach to address these shortcomings and improve overall performance: utilizing a reinforcement deep learning method to carry out online adjustments of parameters in a homogeneous Particle Swarm Optimization, where all particles exhibit identical search behaviors inspired by models of social influence among uniform individuals. The present method utilizes an online parameter control to analyze and adjust each primary PSO parameter, particularly the acceleration factors and the inertia weight. Initially, a partially observed Markov decision process model at the PSO level is used to model the online parameter adaptation. Subsequently, a Hidden Markov Model classification, combined with a Deep Q-Network, is implemented to create a novel Particle Swarm Optimization named DPQ-PSO, and its parameters are adjusted according to deep reinforcement learning. Experiments on different benchmark unimodal and multimodal functions demonstrate superior results over most state-of-the-art methods regarding solution accuracy and convergence speed.

Keywords: particle swarm optimization; partially observed Markov decision process; metaheuristics control; parameter adaptation; deep reinforcement learning; hidden Markov model; deep Q-network



Citation: Aoun, O. Deep Q-Network-Enhanced Self-Tuning Control of Particle Swarm Optimization. *Modelling* **2024**, *5*, 1709–1728. <https://doi.org/10.3390/modelling5040089>

Academic Editor: Ivan Dimov

Received: 25 September 2024
Revised: 4 November 2024
Accepted: 12 November 2024
Published: 15 November 2024



Copyright: © 2024 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Background

Particle Swarm Optimization (PSO), similar to other non-exhaustive optimization methods such as brute-force search [1], often performs well in some problems but fails in others due to the common issue of becoming trapped in local optima or suboptimal solutions. Two primary disadvantages of PSO are premature convergence and dependency on parameter settings. Premature convergence occurs when swarm particles converge too quickly towards a point near the best-known positions, which may not necessarily represent the optimal solution [2]. The rapid information exchange among particles often intensifies this issue, leading to uniformity, reduced diversity, and an increased risk of settling in local optima [3]. Additionally, PSO's performance can vary significantly depending on its parameter settings, which are not universally effective across different problems [4]. The main issue comes from balancing exploration (global search) and exploitation (local search). Multiple methods have been proposed to enhance PSO's effectiveness and reduce its tendency to become stuck in undesirable solutions. The three primary strategies that have been identified for enhancing Particle Swarm Optimization (PSO) are parameter adjustments, modifications to algorithm components, and hybridization with other algorithms.

Adjusting parameters entails customizing several elements of PSO, including either the topology or the significant parameters, such as the weight of inertia, coefficients of acceleration, and the size of the population [5]. Modifying components pertains to altering or updating rules for velocity or position (this may also include introducing new components

or changing how they are calculated). Hybridizing the algorithm involves combining PSO with different techniques to leverage the strengths of multiple approaches. For instance, integrating PSO with clustering algorithms can enhance the optimization process, enabling more effective search techniques [6]. Additionally, incorporating crossover operators from genetic algorithms into PSO may further strengthen the optimization framework [7]. This research specifically examines the integration of machine learning to improve the predictive capabilities of PSO's search process.

Through hybridization with other machine learning algorithms, this paper proposes a novel way of adjusting key PSO parameters, precisely inertia weight and acceleration coefficients. However, selecting optimal parameters is inherently complex and may vary from one problem to another [4].

To address this matter, we model the online parameter setting problem of PSO using a partially observed Markov decision process (POMDP). This model reflects dynamic state changes in particles across different phases: exploration, exploitation, convergence, and transitions out of local optima. Furthermore, the behavior of particles across iterations is monitored through partial observations of these states, which guide the model in selecting the most appropriate action for each belief state. The solution to this model involves both a Hidden Markov Model (HMM) and a Deep Q-Network (DQN). The HMM employs a Viterbi classification algorithm to capture the belief states of the particles. Subsequently, based on deep reinforcement learning techniques, the optimal actions for adjusting PSO parameters—namely the inertia weight and acceleration factors—are determined and applied at each iteration.

In our earlier work [8], we already explored the use of the Hidden Markov Model (a supervised learning technique) for the online estimation and adjustment of parameters in Particle Swarm Optimization (PSO). Building on this, we now advance our approach by integrating a partially observed Markov decision process (POMDP), further enhanced by adding a Deep Q-Network (DQN) specifically to resolve the POMDP. This refined strategy enables dynamic, real-time optimization of PSO parameters with each iteration, offering a more precise and adaptable mechanism for parameter tuning.

The subsequent sections of this research work are arranged as outlined below: Section 2 reveals a comprehensive review of the literature. Section 3 elaborates on the POMDP model and details the integration of the DQN model. Section 4 is entirely devoted to presenting the empirical findings. Finally, the conclusion is provided in Section 5, which encapsulates our findings and reflections.

2. Related Works

Several strategies have been established to improve Particle Swarm Optimization (PSO) in the past decade. Improvements in PSO have been categorized into three key methodologies, as shown in Figure 1: parameter optimization [9], algorithmic component adjustments [10], and integration with other algorithms (hybridization) [11]. Parameter optimization entails fine-tuning PSO settings, including topology, coefficients for acceleration and inertia, and population size. Adjustments to algorithmic components involve modifying velocity or position update rules, which may include creating or recalibrating existing elements. Hybridization with other techniques allows PSO to leverage complementary algorithms, enhancing its performance and problem-solving capabilities.

Firstly, parameter setting for PSO algorithm has caused significant difficulty in the area of iterative optimization techniques in recent years [12]. Recent research [13,14] has identified two primary approaches to parameter setting: parameter tuning and parameter control.

Parameter tuning entails establishing the algorithm's parameters to specific values found via simulations [15]. This approach has also been applied in airline scheduling problems [16,17], where a Hidden Markov Model was investigated for tuning metaheuristics [18]. In contrast, parameter control refers to the process of dynamically modifying

parameter values while the algorithm is running [19,20]. This method can be classified into static, adaptive, or self-adaptive approaches.

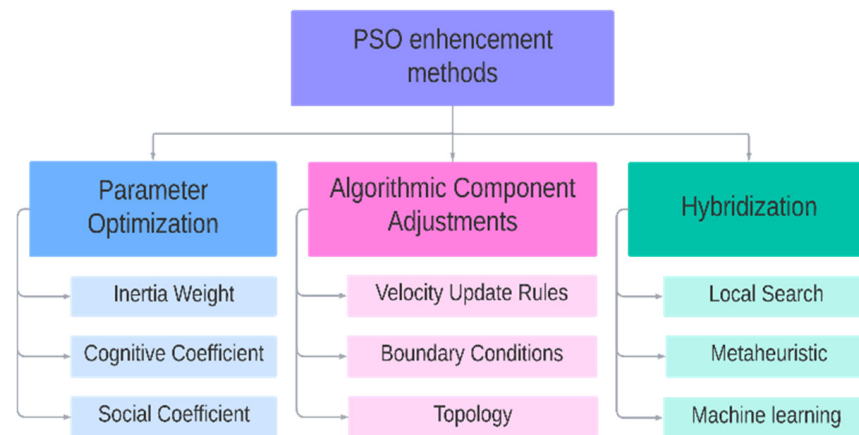


Figure 1. Improvement methods to enhance PSO.

Static parameter control utilizes some pre-established rules, sometimes referred to as a time-varying rule, to modify parameter values depending on the PSO iteration number [21]. Adaptive control of parameters employs some function to establish a relationship between the feedback obtained from the current optimization process and the value of the PSO parameter [22].

In a previous work [8], a Hidden Markov Model (HMM) was used as an online classification method to estimate PSO states, which are exploration, exploitation, convergence, and jumping out [23]. The earlier approach adaptively adjusts the acceleration factors $c1$ and $c2$. In [9], an equivalent HMM-based approach was applied to adapt the population size along with the acceleration factors. Another work [24] leveraged HMM-detected PSO states to control the inertia weight also using HMM classification.

Self-adaptive parameter control embeds the parameters within each particle, enabling these parameters to vary and develop during the algorithm's execution [25]. Notably, the use of a classification model has been explored for adapting PSO parameters, providing a probabilistic framework for dynamic adjustments, like in [26], where the authors proposed a probabilistic finite state machine design for self-parameter adaptations of each particle in PSO, enhancing its adaptability and robustness. Similarly, the authors of [23] further incorporated a self-dynamic adaptation of population size across iterations.

Additionally, PSO parameter adjustment may be performed in two contexts: homogeneous and heterogeneous swarms [27]. In a homogeneous swarm, all particles exhibit uniform behavior, but in a heterogeneous swarm, several distinct behaviors coexist concurrently [28]. Recent studies have introduced cooperative multi-swarm strategies and adaptive cooperation using a Markov Model, enhancing the performance and robustness of PSO in various applications [29,30].

Furthermore, the authors of [20,31] presented an in-depth investigation of the latest advances in PSO, exploring different parameter control methodologies and real-world applications. The research results jointly give important insights into the techniques and the resulting impacts on the PSO algorithm.

Other recent advancements in Particle Swarm Optimization (PSO) have focused on enhancing key algorithmic components, including velocity update rules, boundary conditions, and topology structures. These components are crucial for the overall performance, stability, and convergence of PSO algorithms. The velocity update calculus in Particle Swarm Optimization (PSO) is essential for managing particle movement and ensuring successful convergence. The authors of [32] presented adaptive velocity update techniques that dynamically alter parameters depending on the optimization process. Their analysis gives insights into how these updates boost PSO's convergence and stability. Meanwhile,

managing boundary conditions efficiently is essential for maintaining the search space's integrity and preventing particles from running away. The authors of [33] addressed unique boundary management approaches that retain particle diversity while preserving search space boundaries. Their thorough review underlines recent measures and their influence on PSO performance.

Additionally, the topology of PSO specifies the structure of communication links within particles, significantly affecting the algorithm's efficiency. The authors of [14] examined different topology patterns and their effects on PSO achievement. Their work offers suggestions for choosing suitable structures based on specific optimization problems. Also, the authors of [29,30] examined cooperative plans through PSO topologies, emphasizing the advantages of multi-swarm cooperation and adaptive communication frameworks. These studies indicate how cooperative strategies may boost PSO performance by enhancing exploration and exploitation balance and strengthening the robustness of the optimization process.

Hybridization in PSO leverages multiple strategies to increase the algorithm's effectiveness. The authors of [34] reported that hybrid approaches have shown success in resolving complex optimization problems and therefore validated the usefulness of combining PSO with local search methods, meta-heuristics, and machine learning. One hybrid strategy involves adding local search methods to refine solutions near potential regions determined by PSO, which aids in attaining quicker convergence and more accurate solutions. For example, the authors of [35] combined dynamic multi-PSO with gravitational search algorithms to solve complex optimization problems. Another hybrid approach involves combining PSO with other metaheuristics like simulated annealing or a genetic algorithm, balancing their strengths to avoid local optima and the premature convergence phenomenon. The authors of [36] demonstrated a hybrid PSO–firefly algorithm to enhance cloud performances. The authors of [37] proved that integrating the global search capacity of genetic algorithms with the convergence speed of PSO will result in a significant improvement compared to other methods.

Integrating machine learning with PSO enables adaptive parameter control and dynamic modification of the algorithm's behavior. Techniques like supervised learning [38] and deep learning [39] have been utilized to inform and optimize PSO's parameter adjustments effectively. For instance, the authors of [40] used a multilayer PSO with ANN, adapting network topology and synaptic weights. This adaptive approach allowed for the dynamic control of parameters, boosting optimization efficiency and accuracy. The authors of [41] proposed one reinforcement learning-based parameter adaptation approach for PSO, showcasing its effective and adaptive capabilities. This approach was further validated in a recent study [42], emphasizing the usefulness of reinforcement learning hybridized with PSO for suitable adaptation. Such models can predict optimal settings, assess the optimization state, and guide the search process, significantly enhancing PSO's efficiency [43,44]. They also handle complex optimization challenges, solidifying PSO's role as an effective and versatile hybrid optimization method.

3. Deep Q-Network-Based Adaptive PSO

This section describes the background knowledge, some essential concepts of the literature, and our previous algorithm, HMM-APSO [8], which provided the foundation for presenting our parameter adaptation control method using a Q-network. The suggested technique is about dynamically balancing the states of PSO to identify optimal parameter values which assure success in different optimization settings. Using a Hidden Markov Model (HMM) helps identify these appropriate states via analyzing the evolution and transitions all over the PSO iterations. Once the most suitable state is determined, establishing a Deep Q-Network (DQN) permits the selection of the optimal action, indicating the best suited parameter values in that state. This combination provides adaptive and strategic adjustments to PSO parameters, boosting performance by dynamically modifying

the algorithm. This dynamic adaptation helps strike a balance between PSO states, leading to higher convergence and improvements in accuracy.

3.1. Theoretical Framework

Regarding the new approach proposed this work, we utilize the classical version of Particle Swarm Optimization (PSO) given in [45], which features a global topology where each particle is connected to every other particle and influenced by the global best (gBest) particle. Each particle i is described by two vectors: velocity vector v_i and position vector x_i . The following equations define how those vectors are updated at each iteration t :

$$v_i(t+1) = wv_i(t) + c_1r_1(pBest_i - x_i(t)) + c_2r_2(gBest - x_i(t)) \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

We assume that there are N particles of the swarm in S space. Here, i indicates the particle's index, and t is the iteration's index; r_1 and r_2 are defined in the interval $[0,1]$, representing two independently and uniformly distributed random variables. The inertia weight w is generally adjusted to decrease linearly from 1 to 0 throughout the execution. The constants c_1 and c_2 are named acceleration factors. $pBest_i$ indicates the personal best position of particle i . $gBest$ refers to the global best position.

In Figure 2, the diagram shows the iteration steps of PSO. The stopping criteria commonly have a maximum number of loops or some convergence criteria.

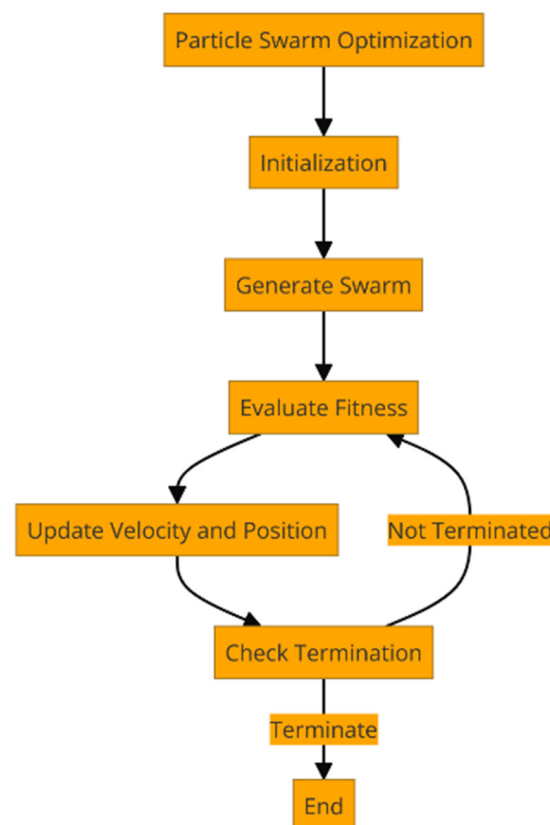


Figure 2. Diagram of PSO iterations.

3.2. Markov Chain on PSO

PSO can be analyzed by examining its stochastic behavior as a multi-stochastic process. As identified by previous researchers, notable work on PSO is based on empirical studies using simulations, and less work has been conducted to analyze PSO theoretically. In [46], a few theoretical propositions were utilized to explore the stochastic process of PSO.

According to [47], the PSO state takes into account as much detail as possible in the process. Previous researchers [46] have proven that the PSO state is memory-less. The state was defined by

$$ST(t) = (X(t), pbest(t), V(t), gbest(t)) \quad (3)$$

W is called a state at time t . It proved the stationarity of the Markov chain on the PSO. We assume that the information contained in $W(t)$ is enough for future moves, and it depends only on the actual iteration state and not the past iteration state. The effect of the current state on the future states is not dependent on its past states. Thus, PSO's behavior only depends on the actual state, not on the succession of the past, which rides on previous achievements.

To evaluate the success of the state, ST , in PSO, an index is defined to reflect the current accomplishment based on particle positions and probability concepts due to their stochastic movements. These states are categorized into classes identified as levels. The levels follow a stochastic process described as

$$\{L(ST(t)), t = 1, 2 \dots\} \quad (4)$$

Forming a Markov chain [48–50] on PSO levels, this approach, detailed in [46], ensures the achievement is position-dependent, aligning with the probabilistic nature of particle movements.

We define, as shown in [51], four evolutionary levels considered as a global state of the PSO swarm, namely the following:

$$L = \{\text{exploration, exploitation, convergence, jumping out}\}.$$

The Markov chain represents the PSO states in Figure 3. The arrows show the possible transitions between states.

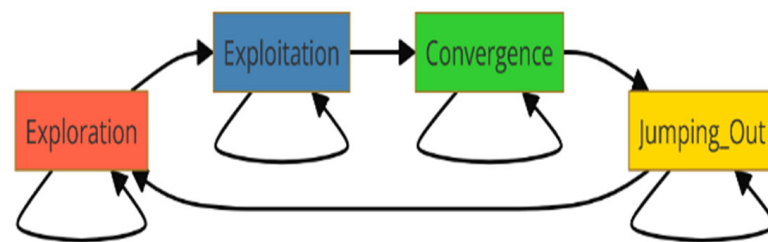


Figure 3. Markov chain on PSO states.

Although the exploration state describes the process of looking at a large area in solution space to avoid local optima, in the exploitation state, the swarm adjusts the solution at the best-known locations to serve the purpose of accuracy. The convergence state involves stable particles, and it is focused on the optimal solution so that the best results may be reached. The jumping-out state adds randomness to escape local optima and continue seeking the global optimum, maintaining variety and avoiding premature convergence. One must balance these states of PSO, giving the appropriate parameter setting for it to be successful when used in optimization situations.

3.3. Partially Observed Markov Decision Process in PSO

Based on the previous paragraph, we define a POMDP [52] on the PSO to build a model of control and adaptation of PSO parameters according to the swarm state. We can characterize a POMDP over PSO since there are already PSO states in the swarm: $L = \{\text{exploration, exploitation, convergence, jumping out}\}$. The state of each particle is not directly observable. Still, it is inferred through an evolutionary factor f that reflects the relative positions of particles and is defined by the average distance of each particle to all others, as described in [51]. The actions are variations in the PSO parameters, and the reward is the measured enhancement of the best solution. This approach gives a perfect

way for PSO to dynamically control and optimize its parameters based on the observed swarm behavior.

Formally, the POMDP is defined as a tuple (S, A, T, Z, R, b_0) as follows:

- States (S)

$$S = \{ s \mid s \in L \} \quad (5)$$

- Observations (O)

$$O = \{ o \mid o = f(Gbest) \} \quad (6)$$

$f(Gbest)$ is the mean distance of Gbest to all other particles:

$$f(Gbest) = \left| \frac{d_{gbest} - d_{min}}{d_{max} - d_{min}} \right| \in [0, 1] \quad (7)$$

where d_i is the distance between a particle i and the other particles.

- Actions (A):

$$A = \{ a \mid a \text{ is a combination of PSO parameters settings} \}$$

We define four parameters setting combinations $\{A\}_1^4$.

Actions include setting values for inertia weight, cognitive coefficient, social coefficient, and randomness.

- Transition Model (T):

$$T(s, a, s') = P(s' \mid s, a) \quad (8)$$

T is the probability of transitioning from state s to state s' given an action a .

- Observation Model (Z):

$$Z(st, a, o) = P(o \mid st, a) \quad (9)$$

Z is the observation probability of o given the next state s' and the action a .

- Reward Function (R):

$$R(s, a) \quad (10)$$

This is the reward received after taking action a in a state s and the measurement enhancement of the best solution.

- Initial Belief State b_0 : This is the initial probability distribution over states. It will be given as an exploration state with a probability equal to 1 and 0 for other states; this means that we assume an exploration state at iteration one.
- The objective is to select the actions $a \in A$ that maximize the expected cumulative reward over time, accounting for the partially observable nature of the states through the observed parameter f .

Then, the policy π maps the history of observations and actions to actions such that the expected sum of rewards is maximized.

To solve this model, we conduct an approach integrating the Hidden Markov Model and Q-network [53]. It is an advantageous solution strategy due to the complementary strengths of these methods.

Using this approach, we can handle partial observability efficiently by using HMMs to maintain and update the belief state of the swarm. A belief state is a probability distribution

over possible states that enables the swarm to make effective inferences about the state of its environment, even when the true state is not directly observable. It combines the powerful deep-learning approximation algorithms of complex functions for high-dimensional state spaces into the Q-network [53] used for action selection. This way, the agent can learn an optimal policy that maximizes the long-term reward while solving this partial observability problem. The state estimate generated by our proposed solution is effectively aggregated by HMMs, leading to scalable Q-networks that can be efficiently used for policy learning in POMDPs.

The following paragraphs will detail the HMM and Q-network models.

3.4. HMM Belief State Classification

The Hidden Markov Model (HMM) is used for PSO state classification due to its advantage in sequence analysis, notably in voice recognition and classification domains. HMMs have proven effectiveness for sequence analysis [54], especially in representing systems with hidden states and time-dependent interactions. This results from their capacity to manage sequential data with inherent stochastic processes. Algorithms including Baum–Welch and Viterbi allow HMMs to continually estimate model parameters while identifying the most probable sequence of hidden states, thereby enhancing accurate classification and prediction in intricate temporal patterns.

We set the HMM with a triple (Π, A, B) , and (Ω, F, P) is a probability space where the whole processes are defined:

- $\Pi = (\pi_i)$, the vector representing initial state probabilities. $\Pi = [1 \ 0 \ 0 \ 0]$: Initial state probability specifying a first deterministic initialization in the exploration state.
- $A = (a_{ij})$, the transition matrix between states, $P(X_t = i | X_{t-1} = j)$, $i, j \in [1, N]$. We supposed a 1/2 value of transition probabilities for all possible transitions in Figure 3.
- $B = (b_{jk})$, the emission matrix named likewise a confusion matrix, $P(Y_t = k | X_t = j)$, $j \in [0, N], k \in [0, M]$.

The evolutionary factor f , representing particle distribution in a search, is used as an observation source in the Hidden Markov Model. Since f is continuous within $[0, 1]$, it is discretized into seven subintervals [23]:

$$[0, 0.2), [0.2, 0.3), [0.3, 0.4), [0.4, 0.6), [0.6, 0.7), [0.7, 0.8), [0.8, 1] \quad (11)$$

Each value of f is assigned to an interval, resulting in a discrete state based on the interval number. Then, matrix B has dimensions of 4×7 .

Probabilities are deduced from an earlier work [23] as follows:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0.5 & 0.25 & 0.25 & 0 \\ 0 & 0.25 & 0.25 & 0.5 & 0 & 0 & 0 \\ \frac{2}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} \end{bmatrix} \quad (12)$$

where once the parameters of the HMM are initialized, the Algorithm 1 Baum–Welch algorithm [55] is employed to compute and update the emission and transition probabilities iteratively. This process enhances the accuracy and adaptability of the HMM during the classification stage.

Algorithm 1: Baum–Welch Algorithm [55]

```

1: Data:  $\Pi, O, A, B, S, Y$ 
2: Initialisation of forward[4,  $T$ ] and backward[4,  $T$ ] matrix
3: repeat
4:   for  $i = 1$  to 4 do
5:      $\text{forward}[S_i, 1] \rightarrow \pi_i \times b_{i,1}$ ;
6:   end
7:   for  $t = 2$  to  $T$  do
8:     for  $i = 1$  to 4 do
9:        $\text{forward}[S_i, t] \rightarrow \sum_{k=1, \dots, 4} \text{forward}[i, t-1] \times a_{k,i} \times b_{i,y_t}$ 
10:    end
11:  end
12:   $\text{forward}[Y_t, T] \rightarrow \sum_{k=1, \dots, 4} \text{forward}[k, T]$ 
13:   $\text{backward}[Y_t, T] \rightarrow \sum_{k=1, \dots, 4} \text{backward}[k, T]$ 
14:  for  $t = 1$  to  $T$  do
15:    for  $i = 1$  to 4 do
16:       $\text{backward}[S_i, t] \rightarrow \sum_{k=1, \dots, 4} \text{backward}[i, t-1] \times a_{k,i} \times b_{i,y_t}$ 
17:    end
18:  end
19:  for  $t = 1$  to  $T$  do
20:    for  $i = 1$  to 4 do
21:      for  $j = 1$  to 4 do
22:         $\xi_{i,j}(t) = \frac{\text{forward}[i, t] \times \text{backward}[i, t] \times b_{i,y_t}}{P(O/(A, B, \pi))}$ 
23:      end
24:    end
25:     $\lambda_i(t) = \frac{\text{forward}[i, t] \times \text{backward}[s, t]}{P(O/(A, B, \pi))}$ 
26:  end
27:  for  $i = 1$  to 4 do
28:    for  $j = 1$  to 4 do
29:       $\pi_i = \lambda_1(t), a_{ij} = \frac{\sum_{t=1, \dots, T-1} \xi_{i,j}(t)}{\lambda_i(t)}, b_{ik} = \frac{\sum_{t=1, \dots, T \cap o_t = Q_k} \xi_{i,j}(t)}{\sum_{t=1, \dots, T} \lambda_i(t)}$ 
30:    end
31:  end
32: until no increase of  $P(O/\lambda)$  or no more iterations are possible
33: Result:  $(\pi, A, B)$ 

```

Subsequently, at each iteration, the Viterbi algorithm [56] is utilized to estimate the belief state of the swarm. Algorithm 2 shows the pseudo-code which describes the state sequence Q (where $Q = q_1 q_2 \dots q_T$) assuming a succession of observations O ($O = o_1 o_2 \dots o_T$). Transitions between the four states are adjusted based on the classifications provided by the HMM.

Additionally, for each state of the swarm as determined by the HMM classification, the subsequent actions are determined by the Q-network, as detailed in the following paragraph.

Algorithm 2: Viterbi algorithm [56]

```

1: Data:  $\Pi, O, A, B, S, Y, T$ 
2: Initialisation of Viterbi[4,T] matrix
3: for  $i = 1$  to 4 do
4:    $Viterbi[i, 1] \rightarrow \pi_i \times b_{i,1}$ ;
5:    $State[i, 1] \rightarrow X_1$ ;
6: end
7: for  $t = 2$  to  $T$  do
8:   for  $i = 1$  to 4 do
9:      $Viterbi[i, t] \rightarrow \max_{s'=1,\dots,4} Viterbi[k, t-1] \times a_{k,i} \times b_{i,y_t}$ 
10:     $State[i, t] \rightarrow \operatorname{argmax}_{s'=1,\dots,4} Viterbi[k, t-1] \times a_{k,i}$ ;
11:   end
12: end
13:  $Y_T \rightarrow \operatorname{argmax}_{k=1,\dots,4} Viterbi[k, T]$  ;
14:  $X_T \rightarrow X_{Y_T}$ 
15: for  $t = T, T-1, \dots, 2$  do
16:    $Y_{t-1} = State[Y_t, t]$ 
17: end
18: Result: The state sequence  $Q_z = (S_{Y_1}, \dots, S_{Y_T})$ 

```

3.5. Deep Q-Network-Based Parameter Setting Actions

After determining the belief state of the PSO, the Q-network is used to determine the suitable action that corresponds to the parametric adjustment control of the PSO. The Q-network, specifically the Deep Q-Network (DQN), is a kind of neural network that will be used to approximate the Q-value function in reinforcement learning, giving the optimal action selection strategy.

A formal description of the Q-network's architecture is provided in Figure 3:

- The input layer of four possible values and four dimensions: state 1 $[1, 0, 0, 0]$, state 2 $[0, 1, 0, 0]$, state 3 $[0, 0, 1, 0]$, and state 4 $[0, 0, 0, 1]$. So, we have four neurons, each representing one element of the one-hot encoded state vector. It will receive the state representation of the belief state in the POMDP calculated previously by the HMM classification. Each neuron in this layer corresponds to one element of the state.
- Hidden layers include a fully connected layer with $h1 = 32$ neurons and ReLU activation.
- The output layer of dimension 4 that corresponds to the number of possible actions.

Q-values that represent the expected cumulative reward for each action in the provided state are determined as follows:

$$Q(s, a) = \text{fitness}(gBest_{i-1}) - \text{fitness}(gBest_i) \quad (13)$$

Fitness is the fitness function used in PSO, and i is the iteration number; we are supposed to have a minimization problem.

The pseudo-code of the Algorithm 3 DQN algorithm is as follows:

Algorithm 3: DQN algorithm [57]

```

1: Data:  $D, Q, \theta, N, M, T, C$ 
2: Initialization: replay memory  $D$  to capacity  $N$ , action
   - value function  $Q$  with random weights  $\theta$ , target action
3:
   - value function  $\hat{Q}$  with weights  $\theta^- = \theta$ ,
4:
5: Set epsilon ( $\epsilon$ ) for exploration – exploitation balance
6: for episode = 1,  $M$  do
7:   Initialize sequence  $s_1 = \{s_1\}$  and preprocessed sequence  $\varphi_1 = \varphi(s_1)$ 
8:   for  $t = 1, T$  do
9:     With probability  $\epsilon$  select a random action  $a_t$ 
10:    otherwise select  $a_t = \operatorname{argmax}_a Q^*(\varphi(s_t), a, \theta)$ 
11:    Execute action  $a_t$  in emulator and observe reward  $r_t$  and next state  $s_{t+1}$ 
12:    Set  $s_{t+1} = s_t, a_t, x_t + 1$  and preprocess  $\varphi_{t+1} = \varphi(s_{t+1})$ 
13:    Store transition  $(\varphi_t, a_t, r_t, \varphi_{t+1})$  in  $D$ 
14:    Sample random minibatch of transitions  $(\varphi_j, a_j, r_j, \varphi_{j+1})$  from  $D$ 
15:    Set target  $y_j = \begin{cases} r_t & \text{for terminal } \varphi_{t+1} \\ r_t + \gamma \max_{a'} \hat{Q}(\varphi_{t+1}, a'; \theta^-) & \text{for non-terminal } \varphi_{t+1} \end{cases}$ 
16:    Perform a gradient descent step on  $(y_j - Q(\varphi_j, a_j; \theta))^2$  with respect to  $\theta$ 
17:
18:     $\theta^- \leftarrow \theta$  every  $C$  steps
19:   end for
20: end for
21: Result: Optimal action – value function  $Q$ 

```

Regarding the action set, each of the four actions defines the parameter setting of acceleration coefficients and inertia weight. The parameter adaptation is carried out according to four actions:

- Action 1:

- Random values of inertia weight between w_{min} and w_{max} :

$$w_i = w_{min} + (w_{max} - w_{min}) * rand() \quad (14)$$

$rand()$: the function that generates random values in $[0,1]$.

- Increase c_1 and decrease c_2 .

- Action 2:

- The inertia weight is calculated according to its distance from other particles:

$$w(f) = \frac{1}{1 + 1.5e^{-2.6f}} \in [0.4, 0.9] \forall f \in [0, 1] \quad (15)$$

- Increase c_1 and slightly decrease c_2 .

- Action 3:

- The maximum value of $w = w_{max}$.
- Slightly increase c_1 and decrease c_2 .

- Action 4:

- The minimum value of the inertia weight: $w_i = w_{min}$.
- Decrease c_1 and increase c_2 .

These parameter variation actions are deduced from the best-known literature on PSO parameters' online control, such as [24,51].

The complete applied framework for adapting the parameters in PSO is described in the next paragraph.

3.6. The PSO-Based DQN Algorithm

The parameter adaptation of PSO will be carried out according to the POMDP presented earlier. The POMDP framework enables the modeling of uncertain environments, providing a robust mechanism for dynamically adapting the parameters of PSO based on

the actual state and observations. In this approach, a Hidden Markov Model (HMM) will be used to identify hidden states that correspond to various ways of adapting the optimization process, including exploration, exploitation, jumping out, and convergence. By recognizing these hidden states, the HMM can adequately track the optimization dynamics and provide information for the following actions. The DQN, with its advanced deep learning capabilities, will subsequently choose the most appropriate strategy for selecting actions in each recognized state. The following Algorithm 4 illustrates the approach steps.

Algorithm 4: DQNPSO

```

1: Data: The objective function ( $F$ )
2: Initialization: iteration  $t = 0$ , positions  $X = \{x_i\}$ , velocities  $V = \{v_i\}$ , inertia weight  $w$ ,
3: HMM parameters ( $\Pi, A, B, S, Y$ ), observations  $O$ , states  $Q$ , swarm size  $N$ , DQN Parameters ( $D, Q, \theta, N, M, T, C$ )
4: while (number of iterations  $t \leq tmax$  not met) do
5:   for  $i = 1$  to  $N$  do
6:      $O[t] \leftarrow l_i$  (the observation sequence update)
7:      $(A, B, \Pi) \leftarrow \text{BaumWelch}(\Pi, O, A, B, \Pi, S, Y)$ ; (the HMM probabilities update)
8:      $(Q_i[1], \dots, Q_i[t]) \leftarrow \text{viterbi}(\Pi, O, A, B, \Pi, S, Y)$ ; (State classification)
9:      $a_i \leftarrow \text{DQN algorithm}$  (action to execute)
10:    Switch(learned action  $a_i$ ): -cases from Rules in paragraph 3.4 -
11:    |  $-w \leftarrow \text{updateRule}(w)$ ;
12:    |  $-(c_1, c_2) \leftarrow \text{updateRule}(c_1, c_2)$ ;
13:    |  $-x_i(t) \leftarrow \text{equation.1}(x_i(t-1), v_i(t-1))$ ; (Update positions)
14:    |  $-v_i(t) \leftarrow \text{equation.2}(x_i(t-1), v_i(t-1))$ ; (Update velocities)
15:    | compute  $f(x_i)$ ;
16:    | if ( $f(x_i) \leq fbest$ ) then
17:    | |  $fbest \leftarrow f(x_i)$ ;
18:    | |  $pbest \leftarrow x_i$ ;
19:    | end if
20:    | if ( $f(pbest) \leq fGbest$ ) then
21:    | |  $fGbest \leftarrow fbest$ ;
22:    | |  $gbest \leftarrow Xbest$ ;
23:    | end if
24:    | end for
25:    |  $t \rightarrow t + 1$ ;
26: end while
27: Result:  $pbest$  and  $fbest$  (the best particle and the best fitness)

```

The PSO method is designed to optimize its parameters by maintaining a balance between exploring new solutions and exploiting established, highly effective ones. It is also successful at escaping local optima and rapidly converging to optimum solutions, enhancing overall performance. An experimental study was conducted, and it is described in the following section to display the effectiveness of the newly adopted method.

4. Experimental Study

We empirically evaluated the provided approach for adapting PSO parameters based on HMM and DQN. We simulated several benchmark functions, including unimodal and multimodal categories. Subsequently, the findings are compared with those obtained from other modern PSO variations to evaluate our method's efficiency.

4.1. Parameters Configuration

Several interconnected variants of the PSO algorithm from the literature were chosen for comparison during testing (Table 1).

The suggested DQNPSO approach was simulated and validated on many benchmark functions (see Table 2).

The empirical study was carried out by iteratively conducting simulations ten times, with identical beginning parameter values. The population size was set to 30, with each particle having a dimension of 30. In each execution, a total of 1000 generations were

executed. The inertia weight was set by default to specific settings given in Table 1. The parameters c_1 and c_2 were initialized with the value 2. The learning rate for the DQN was fixed at a value of 0.001, and the number of steps was 10. The tests were executed on a system with a configuration of an Intel i7 10th-generation processor and 16 GB of RAM. We focus on two performance metrics: solution accuracy and convergence speed.

Table 1. The selected PSO variants from the literature.

Algorithm	Name	Specific Settings	Reference
APSO	Adaptive PSO	$c_1 = c_2 = 2, \omega = 0.9$	[51]
LinWPSO	Linear decreasing weights PSO	$\omega_{min} = 0.0001, \omega_{max} = 0.1$	[58]
AsyLnCPSO	Asynchronous PSO	$c_{1min} = c_{2min} = 0.01, c_{1max} = c_{2max} = 2$	[59]
RandWPSO	Random inertia weight PSO	$c_1 = c_2 = 2, \omega_{min} = 0.01, \omega_{max} = 0.9$	[60]
YSPSO	PSO with compressibility factor	$c_1 = c_2 = 2, \omega = 0.9$	[61]
SecVibratPSO	Order oscillating PSO	$c_1 = c_2 = 2, \omega = 0.9$	[62]
CLSPSO	Cooperative line search PSO	$c_1 = c_2 = 2, \omega = 0.9$	[63]

Table 2. Standard test functions.

Function	Name	Category
$f_1 = \sum_{i=1}^D [(10^6)^{\frac{i-1}{D-1}} x_i^2]$	Elliptic	Unimodal
$f_2 = \sum_{i=1}^D (x_i + 0.5)^2$	Step	Unimodal
$f_3 = \sum_{i=1}^D x_i^2$	Sphere	Unimodal
$f_4 = 10^6 x_1^2 + \sum_{i=2}^D x_i^2$	Tablet	Unimodal
$f_5 = \sum_{i=1}^D (\sum_{j=1}^D x_j)^2$	Quadric	Unimodal
$f_6 = \sum_{i=1}^D [x_i^2 - 10 \cos(2\pi x_i) + 10]$	Rastrigrin	Multimodal
$f_7 = -20 \exp(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2})$	Ackley	Multimodal
$f_8 = \frac{1}{4000} \sum_{i=1}^D x_i^2 - \Pi \cos(x_i / \sqrt{i}) + 1$	Griewang	Multimodal
$f_9 = \sum_{i=1}^D x_i \sin(\sqrt{ x_i })$	Schewefel	Multimodal
$f_{10} = -\frac{1 + \cos(12\sqrt{x_1^2 + x_2^2})}{1/2(x_1^2 + x_2^2) + 2}$	Drop wave	Multimodal

4.2. Solution Accuracy

In order to assess the efficacy of our DQNPSO approach, we performed a comparison study by testing it against several PSO variations utilizing benchmark test functions. Each benchmark function was performed over all PSO variations, and the mean and best values were obtained to evaluate the solution accuracy when measured against other PSO variants published in the literature. The findings of this comparison are shown in Table 3.

We can see in Table 3 that DQNPSO enhances PSO when compared to the selected PSO variants from published research. There is a considerable improvement across DQNPSO compared to the other techniques. Our suggested approach delivers much superior results. DQNPSO has developed considerably higher solution accuracy efficiency for unimodal and multimodal functions.

Table 3. Comparative analysis of PSO variant results.

Functions	DQNPSO	PSO	APSO	YSPSO	LinWPSO	CLSPSO	AsyLnCPSO	RandWPSO	SecVibratPSO	
f_1	Best	0.010212	241×10^6	29×10^6	367×10^6	372×10^6	7734×10^6	222×10^6	4451×10^6	911×10^6
	Mean	6.7226	7646×10^6	134×10^6	1074×10^6	624×10^7	292×10^8	1002×10^6	1262×10^6	$22,265 \times 10^6$
f_2	Best	0	9.13×10^{-6}	0	0	0	0	0.00001166	16,867	
	Mean	0	0.21843	0	0	0	11,393	1.41×10^{-25}	23,771	92,849
f_3	Best	7.104×10^{-6}	292,268	48,545	73,068	159,127	716,543	165,327	272,797	46,138
	Mean	0.00175	463,932	24,973	14,448	31,6226	2,259,075	408,782	607,237	647,525

Table 3. Cont.

Functions		DQNPSO	PSO	APSO	YSPSO	LinWPSO	CLSPSO	AsyLnCPSO	RandWPSO	SecVibratPSO
f_4	Best	1.67×10^{-5}	985,342	11,408	220,672	34,463	2,021,904	240,971	1,186,314	3888
	Mean	0.00348	164,255	42,255	418,765	1,054,338	4,028,212	439,076	2,111,731	1,951,189
f_5	Best	0.38443	2.44×10^{11}	871×10^6	170×10^7	3484×10^7	2.050×10^{12}	369×10^8	6.586×10^{11}	1325×10^7
	Mean	624.22	1.61×10^{12}	8934×10^6	6950×10^7	9.927×10^{11}	7.07×10^{12}	1856×10^7	3.105×10^{12}	6.228×10^{12}
f_6	Best	0.000115	2,189,475	373,897	1,099,642	170,471	3,712,168	1,977,706	2,043,068	2,095,081
	Mean	3.6364	3,074,856	573,572	1,821,846	2,583,946	5,351,072	2,905,228	3,133,353	3,280,662
f_7	Best	0.000725	47,818	14,562	3659	44,627	73,411	45,955	48,573	18,349
	Mean	0.020312	56,504	22,541	44,661	54,669	94,979	5951	58,876	51,858
f_8	Best	5.46×10^{-8}	15,527	59,507	71,176	13,023	38,083	68,348	15,657	39,343
	Mean	0.01632	38,999	26,352	16,698	32,301	75,682	19,472	53,535	46,518
f_9	Best	0.012693	572,305	33,239	155,126	325,646	1,583,686	186,099	68,254	108,678
	Mean	3.6911	995,474	563,267	247,067	663,407	4,052,457	587,928	1,769,817	2,338,293
f_{10}	Best	-1	-99,999	-1	-1	-1	-1	-1	-99,297	-9943
	Mean	-1	-9599	-9856	-98,972	-96,298	-94,403	-99,383	-94,333	-82,273

4.3. Process Time

We analyzed the process time of the DQNPSO approach against other PSO variants based on the CPU execution time for all benchmark functions. Since the execution is performed solely on the CPU without any parallelism or GPU implementation, our focus is exclusively on the CPU time.

The execution durations of several PSO variants are illustrated in Figure 4. It emphasizes that DQNPSO has execution durations that are substantially longer and more variable than other PSO variants. This is a result of the use of a Deep Q-Network (DQN), which incorporates complex neural network computations. The execution durations of the other PSO variants, including APSO, PSO, LinWPSO, and others, are consistently lower and more predictable, suggesting more efficient and reliable performance. This implies that DQNPSO operates at the expense of computational efficacy. However, the high accuracy performance demonstrated in the previous paragraph can largely compensate for this time consumption. In addition, this issue of CPU time consumption can be addressed, and we can implement GPU execution and incorporate parallelism capabilities into our algorithm. This approach is supported by studies such as [64].

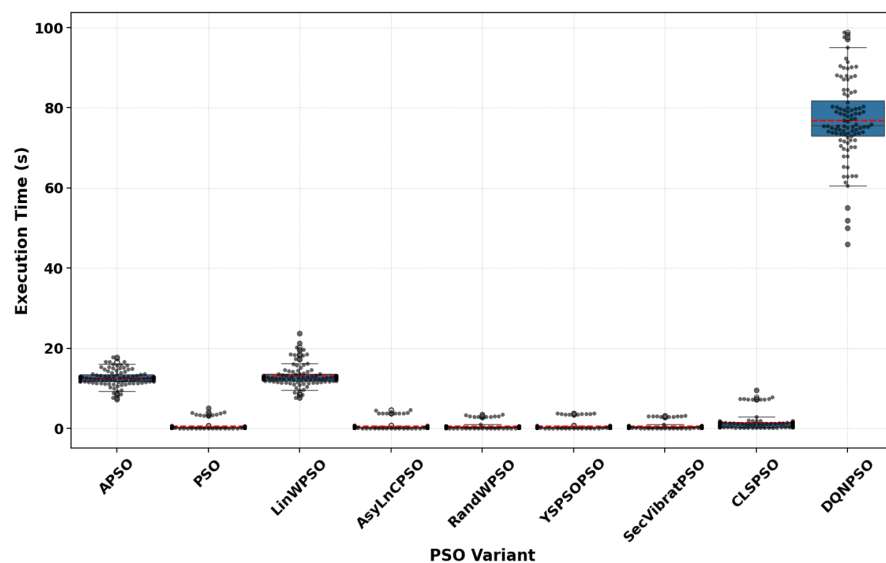


Figure 4. Comparison of execution time in seconds across different PSO variants. Grey dots represent individual execution times, blue boxes show the interquartile range, and red lines indicate mean execution times.

4.4. Convergence Speed

We conducted a comparison regarding the convergence speed for the ten benchmark functions.

The charts used for the comparative analysis of the convergence speed are presented in Figure 5. The red line of the DQNPSO executions' cross-iterations is below all other chart lines. Consequently, DQNPSO delivers a speedier convergence than the previously employed PSO variations in the literature.

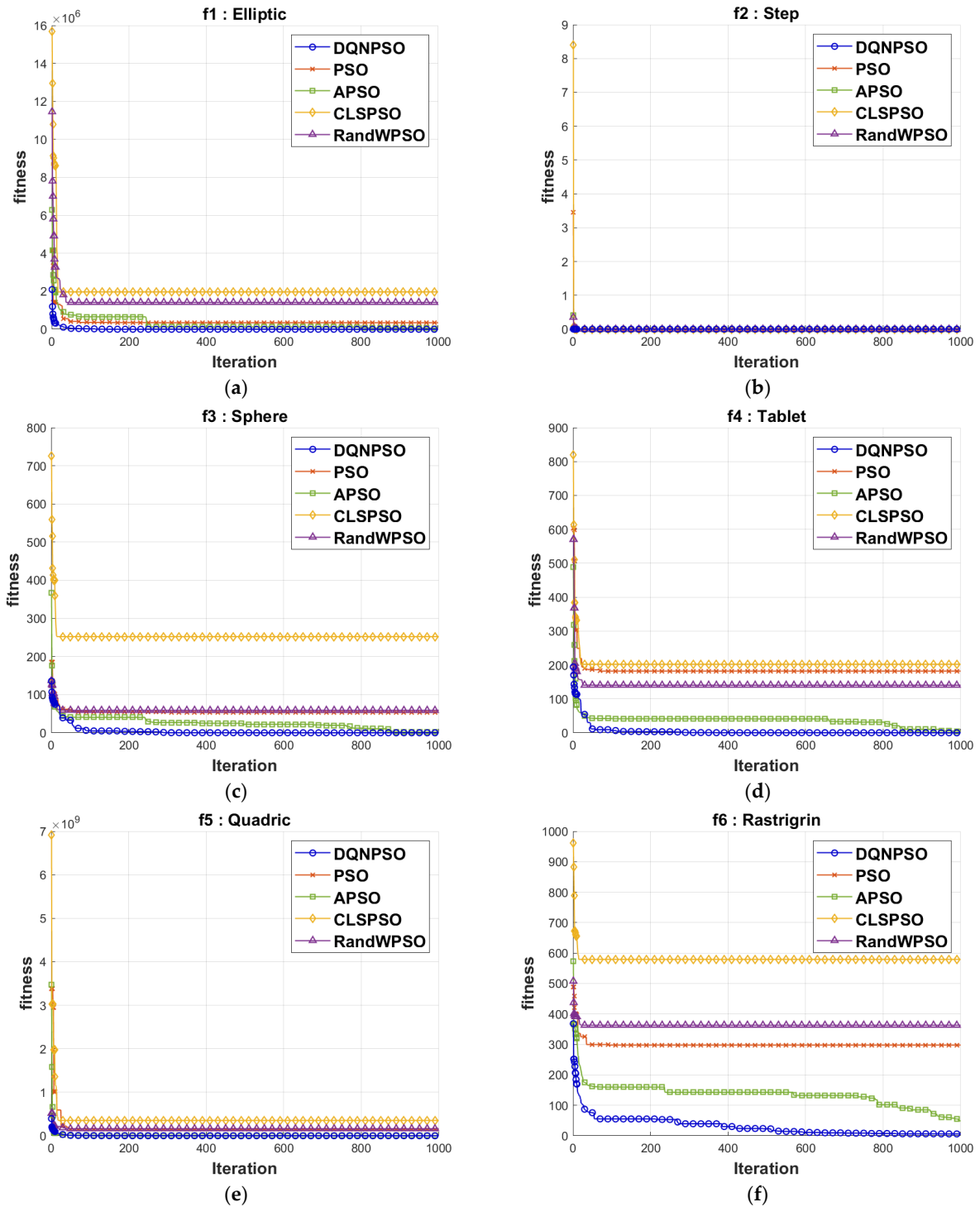


Figure 5. Cont.

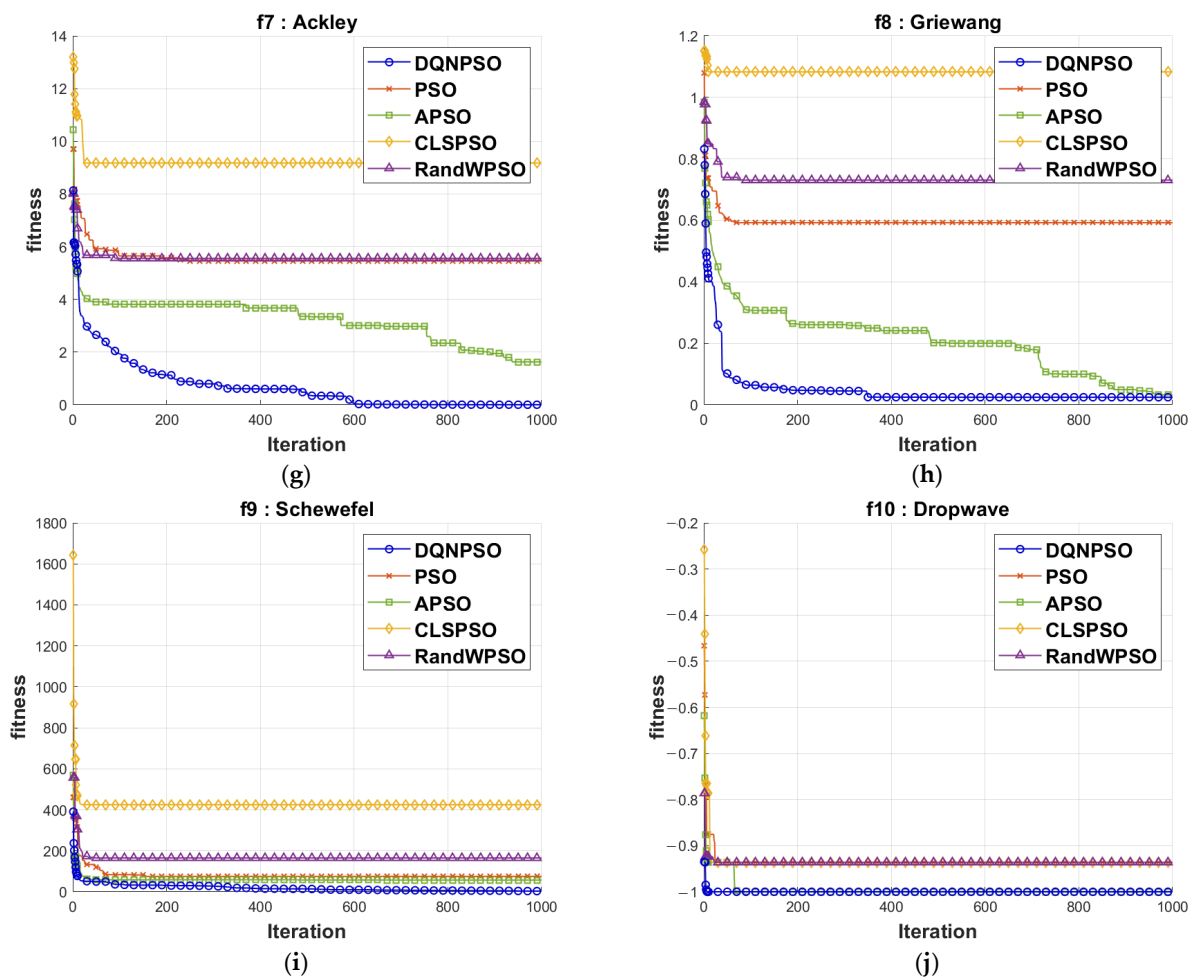


Figure 5. Comparison of convergence speed on benchmark functions for (a) Elliptic; (b) Step; (c) Sphere; (d) Tablet; (e) Quadric; (f) Rastrigrin; (g) Ackley; (h) Griewang; (i) Schewefel; and (j) Drop wave.

4.5. Statistical Test Evaluation

To further compare DQNPSO with other selected PSO algorithms, we employed parametric two-sided testing, specifically the *t*-test, at a 0.05 significant level. Using this method makes it easier to assess whether the observed performance differences are due to chance and whether they are statistically significant [65]. Using the 0.05 criterion enabled us to definitely determine whether DQNPSO exhibits real gains over other PSO algorithms or if the observed changes lack a significant effect. The *t*-test findings define the outcomes as significant (p -value < 0.05), marginally significant ($0.05 \leq p$ -value < 0.1), or not significant (p -value ≥ 0.1), permitting a straightforward evaluation of comparative performance and providing an evidence-based foundation for analysis.

The statistical comparison (see Table 4) of DQN against multiple PSO algorithms indicates that DQN consistently outperforms other PSO versions, with only very few occasions when its performance equals that of the others. This is underscored by the large count of +1 (better) results across all comparisons, where DQN demonstrated superior performance in 90% to 100% of the cases. APso, AsyLnCPSO, RandWPSO, YSPSO, and CLSPSO each presented a single result of 0 (Same), illustrating marginal differences where DQN performed similarly but not worse. Evidently, there were no examples of -1 (worse), indicating DQN’s robustness and better or equivalent performance across all tests. This analysis confirms DQNPSO’s strong potential compared to standard and modified PSO algorithms.

Table 4. Statistical *t*-test comparison of DQNPSO with other PSO algorithms.

	PSO	APSO	LinWPSO	AsyLnCPSO	RandWPSO	YSPSO	SecVibratPSO	CLSPSO
f_1	0	0	0	0	0	0	0	0
f_2	0.0260	0	0	0.0040	0.0797	0	0.0149	0.0710
f_3	0	0	0	0	0	0	0	0
f_4	0	0	0	0	0	0	0	0
f_5	0	0	0	0	0	0	0	0
f_6	0	0	0	0	0	0	0	0
f_7	0	0	0	0	0	0	0	0
f_8	0	0.0129	0	0	0	0	0	0
f_9	0	0	0	0	0	0	0	0
f_{10}	0	0.0822	0	0.0745	0	0.0536	0	0
+1 (Better)	10	9	10	9	9	9	10	9
0 (Same)	0	1	0	1	1	1	0	1
-1 (Worse)	0	0	0	0	0	0	0	0

5. Conclusions

In conclusion, this research proposes significant advances in Particle Swarm Optimization (PSO) by integrating a deep machine learning approach, namely Deep Q-network, for dynamic parameter setting in a homogenous PSO framework. By solving the prevalent issues of slowing down convergence or being stuck in local optima, this suggested approach considerably boosts the entire performance of the PSO. The newly introduced DPQ-PSO framework, which combines a partly observed Markov decision process model with Hidden Markov Model classification and a Deep Q-Network, offers an adaptive method for real-time parameter adaptation. The experimental findings regarding several benchmark unimodal and multimodal functions verify the superior performance of the DPQ-PSO algorithm, providing considerable increases in solution accuracy and convergence speed compared to current techniques. However, this approach suffers from an increased CPU time due to the computational complexity introduced by the integration of deep learning. This novel method not only increases the application capacity of PSO in handling complex optimization issues but also sets a new benchmark in improving metaheuristic algorithms using deep machine learning approaches.

Future research should focus on the augmented computational time resulting from deep learning integration by exploring model optimization strategies, such as pruning the DQN parameters, alongside parallel computing methods for improved scalability. Extending DPQPSO to heterogeneous systems could increase solution variety, while integrating other advanced reinforcement learning techniques like Proximal policy optimization could further develop parameter adaptation. Furthermore, the DPQPSO framework has substantial potential for real-world applications, improving performance in several domains, and implementing this method in complex optimization tasks, such as engineering design and scheduling, will further support its efficacy.

Funding: This research received no external funding.

Data Availability Statement: Related simulation data can be provided upon further inquiry.

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

References

1. Schaeffer, J.; Lu, P.; Szafron, D.; Lake, R. A re-examination of brute-force search. In Proceedings of the AAAI Fall Symposium on Games: Planning and Learning, Raleigh, NC, USA, 22–24 October 1993; pp. 51–58.
2. van den Bergh, F.; Engelbrecht, A. A Cooperative Approach to Particle Swarm Optimization. *IEEE Trans. Evol. Comput.* **2004**, *8*, 225–239. [[CrossRef](#)]
3. Bonyadi, M.R.; Michalewicz, Z. Particle swarm optimization for single objective continuous space problems: A review. *Evol. Comput.* **2017**, *25*, 1–54. [[CrossRef](#)] [[PubMed](#)]
4. Premalatha, K.; Natarajan, A. Hybrid PSO and GA for global maximization. *Int. J. Open Probl. Compt. Math.* **2009**, *2*, 597–608.
5. Liu, H.R.; Cui, J.C.; Lu, Z.D.; Liu, D.Y.; Deng, Y.J. A hierarchical simple particle swarm optimization with mean dimensional information. *Appl. Soft Comput.* **2019**, *76*, 712–725. [[CrossRef](#)]
6. AKOPOV, A.S. A Clustering-Based Hybrid Particle Swarm Optimization Algorithm for Solving a Multisectoral Agent-Based Model. *Stud. Inform. Control* **2024**, *33*, 83–95. [[CrossRef](#)]
7. Chen, Y.; Li, L.; Xiao, J.; Yang, Y.; Liang, J.; Li, T. Particle swarm optimizer with crossover operation. *Eng. Appl. Artif. Intell.* **2018**, *70*, 159–169. [[CrossRef](#)]
8. Aoun, O.; Sarhani, M.; El Afia, A. Hidden markov model classifier for the adaptive particle swarm optimization. In *Recent Developments in Metaheuristics*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 1–15. [[CrossRef](#)]
9. Aoun, O.; Sarhani, M.; Afia, A.E. Particle swarm optimisation with population size and acceleration coefficients adaptation using hidden Markov model state classification. *Int. J. Metaheuristics* **2018**, *7*, 1–29. [[CrossRef](#)]
10. Khursheed, A.; Ilyas, M.; Rafi, K.; Azad, A. A Novel Modified PSO Algorithm to Optimise the PV Output Power of Grid-Connected PV System. *Int. J. Electr. Electron. Eng.* **2023**, *10*, 188–198. [[CrossRef](#)]
11. Wang, Y.; Habib, K.; Wadood, A.; Khan, S. The Hybridization of PSO for the Optimal Coordination of Directional Overcurrent Protection Relays of the IEEE Bus System. *Energies* **2023**, *16*, 3726. [[CrossRef](#)]
12. Zhang, J.; Zhan, Z.; Lin, Y.; Chen, N.; Gong, Y.; Zhong, J.; Chung, H.S.; Li, Y.; Shi, Y. Evolutionary computation meets machine learning: A survey. *IEEE Comput. Intell. Mag.* **2011**, *6*, 68–75. [[CrossRef](#)]
13. Chen, F.; Sun, X.; Wei, D.; Tang, Y. Tradeoff strategy between exploration and exploitation for PSO. In Proceedings of the 2011 Seventh International Conference on Natural Computation, Shanghai, China, 26–28 July 2011; Volume 3, pp. 1216–1222. [[CrossRef](#)]
14. Shami, T.M.; El-Saleh, A.A.; Alswaitti, M.; Al-Tashi, Q.; Summakieh, M.A.; Mirjalili, S. Particle swarm optimization: A comprehensive survey. *IEEE Access* **2022**, *10*, 10031–10061. [[CrossRef](#)]
15. Zhang, Y.; Wang, S.; Ji, G. A comprehensive survey on particle swarm optimization algorithm and its applications. *Math. Probl. Eng.* **2015**, *2015*, 931256. [[CrossRef](#)]
16. Aoun, O.; El Afia, A. Time-Dependence in Multi-Agent MDP Applied to Gate Assignment Problem. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 331–340. [[CrossRef](#)]
17. El Afia, A.; Aoun, O. Data-driven based aircraft maintenance routing by markov decision process model. In *Proceedings of the 2nd International Conference on Big Data, Cloud and Applications*; New York, NY, USA, 29–30 March 2017, BDCA'17. [[CrossRef](#)]
18. Aoun, O.; Sarhani, M.; El Afia, A. Investigation of hidden markov model for the tuning of metaheuristics in airline scheduling problems. *IFAC-Pap.* **2016**, *49*, 347–352. [[CrossRef](#)]
19. Nguyen, T.T.; Yang, S.; Branke, J. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm Evol. Comput.* **2012**, *6*, 1–24. [[CrossRef](#)]
20. Jain, M.; Saihjpal, V.; Singh, N.; Singh, S.B. An Overview of Variants and Advancements of PSO Algorithm. *Appl. Sci.* **2022**, *12*, 8392. [[CrossRef](#)]
21. Romasevych, Y.; Loveikin, V.; Loveikin, Y.V. Development of a PSO Modification with Varying Cognitive Term. In Proceedings of the 2022 IEEE 3rd KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 3–7 October 2022; pp. 1–5. [[CrossRef](#)]
22. Xu, G.; Cui, Q.; Shi, X.; Ge, H.; Zhan, Z.H.; Lee, H.P.; Liang, Y.; Tai, R.; Wu, C. Particle swarm optimization based on dimensional learning strategy. *Swarm Evol. Comput.* **2019**, *45*, 33–51. [[CrossRef](#)]
23. Aoun, O.; El Afia, A.; Garcia, S. Self Inertia Weight Adaptation for the Particle Swarm Optimization. In Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications, Rabat, Morocco, 2–5 May 2018; Volume 8, pp. 1–6.
24. El Afia, A.; Sarhani, M.; Aoun, O. Hidden markov model control of inertia weight adaptation for Particle swarm optimization. *IFAC-Pap.* **2017**, *50*, 9997–10002. [[CrossRef](#)]
25. Isiet, M.; Gadala, M. Self-adapting control parameters in particle swarm optimization. *Appl. Soft Comput.* **2019**, *83*, 105653. [[CrossRef](#)]
26. El Afia, A.; Sarhani, M.; Aoun, O. A Probabilistic Finite State Machine Design of Particle Swarm Optimization. *Stud. Comput. Intell.* **2019**, *774*, 185–201. [[CrossRef](#)]
27. Varna, F.T.; Husbands, P. HIDMS-PSO: A New Heterogeneous Improved Dynamic Multi-Swarm PSO Algorithm. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, Australia, 1–4 December 2020; pp. 473–480. [[CrossRef](#)]
28. Ye, W.; Feng, W.; Fan, S. A novel multi-swarm particle swarm optimization with dynamic learning strategy. *Appl. Soft Comput.* **2017**, *61*, 832–843. [[CrossRef](#)]

29. El Afia, A.; Aoun, O.; Garcia, S. Adaptive cooperation of multi-swarm particle swarm optimizer-based hidden Markov model. *Prog. Artif. Intell.* **2019**, *8*, 441–452. [[CrossRef](#)]
30. Aoun, O.; El Afia, A.; Talbi, E.G. A Cooperative Multi-swarm Particle Swarm Optimizer Based Hidden Markov Model. *Stud. Comput. Intell.* **2021**, *906*, 315–334. [[CrossRef](#)]
31. Liu, J.; Sarker, R.; Elsayed, S.; Essam, D.; Siswanto, N. Large-scale evolutionary optimization: A review and comparative study. *Swarm Evol. Comput.* **2024**, *85*, 101466. [[CrossRef](#)]
32. Khan, T.A.; Ling, S.; Mohan, A. Advanced Particle Swarm Optimization Algorithm with Improved Velocity Update Strategy. In Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 7–10 October 2018; pp. 3944–3949. [[CrossRef](#)]
33. Gandomi, A.; Kashani, A. Probabilistic evolutionary bound constraint handling for particle swarm optimization. *Oper. Res.* **2018**, *18*, 801–823. [[CrossRef](#)]
34. Thangaraj, R.; Pant, M.; Abraham, A.; Bouvry, P. Particle swarm optimization: Hybridization perspectives and experimental illustrations. *Appl. Math. Comput.* **2011**, *217*, 5208–5226. [[CrossRef](#)]
35. Nagra, A.A.; Han, F.; Ling, Q.; Mehta, S. An Improved Hybrid Method Combining Gravitational Search Algorithm With Dynamic Multi Swarm Particle Swarm Optimization. *IEEE Access* **2019**, *7*, 50388–50399. [[CrossRef](#)]
36. Lilhore, U.K.; Simaiya, S.; Maheshwari, S.; Manhar, A.; Kumar, S. Cloud performance evaluation: Hybrid load balancing model based on modified particle swarm optimization and improved metaheuristic firefly algorithms. *Int. J. Adv. Sci. Technol.* **2020**, *29*, 12315–12331.
37. Shao, K.; Song, Y.; Wang, B. PGA: A New Hybrid PSO and GA Method for Task Scheduling with Deadline Constraints in Distributed Computing. *Mathematics* **2023**, *11*, 1548. [[CrossRef](#)]
38. Jin, H.; Kim, Y.G.; Jin, Z.; Rushchitc, A.A.; Al-Shati, A.S. Optimization and analysis of bioenergy production using machine learning modeling: Multi-layer perceptron, Gaussian processes regression, K-nearest neighbors, and Artificial neural network models. *Energy Rep.* **2022**, *8*, 13979–13996. [[CrossRef](#)]
39. Chaganti, R.; Mourade, A.; Ravi, V.; Vemprala, N.; Dua, A.; Bhushan, B. A Particle Swarm Optimization and Deep Learning Approach for Intrusion Detection System in Internet of Medical Things. *Sustainability* **2022**, *14*, 12828. [[CrossRef](#)]
40. Lee, K.E.; bin Abdul Aziz, I.; bin Jaafar, J. Adaptive Multilayered Particle Swarm Optimized Neural Network (AMPSONN) for Pipeline Corrosion Prediction. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 499–508. [[CrossRef](#)]
41. Wu, D.; Wang, G.G. Employing reinforcement learning to enhance particle swarm optimization methods. *Eng. Optim.* **2022**, *54*, 329–348. [[CrossRef](#)]
42. Yin, S.; Jin, M.; Lu, H.; Gong, G.; Mao, W.; Chen, G.; Li, W. Reinforcement-learning-based parameter adaptation method for particle swarm optimization. *Complex Intell. Syst.* **2023**, *9*, 5585–5609. [[CrossRef](#)]
43. Zhang, F.; Chen, Z. A Novel Reinforcement Learning-Based Particle Swarm Optimization Algorithm for Better Symmetry between Convergence Speed and Diversity. *Symmetry* **2024**, *16*, 1290. [[CrossRef](#)]
44. Huang, W.; Liu, Y.; Zhang, X. Hybrid Particle Swarm Optimization Algorithm Based on the Theory of Reinforcement Learning in Psychology. *Systems* **2023**, *11*, 83. [[CrossRef](#)]
45. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the IEEE International Conference Neural Networks, Perth, Australia, 27 November–1 December 1995; pp. 1942–1948.
46. Chou, C.W.; Lin, J.H.; Yang, C.H.; Tsai, H.L.; Ou, Y.H. Constructing a Markov Chain on Particle Swarm Optimizer. In Proceedings of the 2012 Third International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA), Kaohsiung, Taiwan, 26–28 September 2012; pp. 13–18.
47. Chou, C.W.; Lin, J.H.; Jeng, R. Markov Chain and Adaptive Parameter Selection on Particle Swarm Optimizer. *Int. J. Soft Comput.* **2013**, *4*, 1. [[CrossRef](#)]
48. Aoun, O.; El Afia, A. Aoun, O.; El Afia, A. A robust crew pairing based on Multi-agent Markov Decision Processes. In Proceedings of the Second World Conference on Complex Systems (WCCS), Agadir, Morocco, 10–12 November 2014; pp. 762–768. [[CrossRef](#)]
49. Aoun, O.; El Afia, A. Using Markov decision processes to solve stochastic gate assignment problem. In Proceedings of the International Conference on Logistics and Operations Management (GOL), Rabat, Morocco, 5–7 June 2014; pp. 42–47. [[CrossRef](#)]
50. Aoun, O.; El Afia, A. Application of multi-agent Markov decision processes to gate assignment problem. In Proceedings of the Third IEEE International Colloquium in Information Science and Technology (CIST), Tetouan, Morocco, 20–22 October 2014; pp. 196–201. [[CrossRef](#)]
51. Zhan, Z.H.; Zhang, J.; Li, Y.; Chung, H.H. Adaptive Particle Swarm Optimization. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **2009**, *39*, 1362–1381. [[CrossRef](#)]
52. Kaelbling, L.P.; Littman, M.L.; Cassandra, A.R. Planning and acting in partially observable stochastic domains. *Artif. Intell.* **1998**, *101*, 99–134. [[CrossRef](#)]
53. Hausknecht, M.; Stone, P. Deep recurrent q-learning for partially observable mdps. In Proceedings of the 2015 AAAI Fall Symposium Series, Arlington, VA, USA, 12–14 November 2015.
54. Kong, X.; Liu, X.; Chen, S.; Kang, W.; Luo, Z.; Chen, J.; Wu, T. Motion Sequence Analysis Using Adaptive Coding with Ensemble Hidden Markov Models. *Mathematics* **2024**, *12*, 185. [[CrossRef](#)]
55. Durbin, R. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*; Cambridge University Press: Cambridge, UK, 1998.

56. Rabiner, L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **1989**, *77*, 257–286. [[CrossRef](#)]
57. 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](#)]
58. Sun, S.; Ye, G.; Liang, Y.; Liu, Y.; Pan, Q. Dynamic Population Size Based Particle Swarm Optimization. In *Advances in Computation and Intelligence; Lecture Notes in Computer Science*; Kang, L., Liu, Y., Zeng, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4683, pp. 382–392. [[CrossRef](#)]
59. Zeng, G.; Jiang, Y. A Modified PSO Algorithm with Line Search. In Proceedings of the 2010 International Conference on Computational Intelligence and Software Engineering (CiSE), Wuhan, China, 10–12 December 2010; pp. 1–4. [[CrossRef](#)]
60. Jiang, W.; Zhang, Y.; Wang, R. Comparative study on several PSO algorithms. In Proceedings of the 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, China, 31 May–2 June 2014; pp. 1117–1119. [[CrossRef](#)]
61. Liu, L.; Gao, X. An adaptive simulation of bacterial foraging algorithm. *Basic Sci. J. Text. Univ.* **2012**, *4*, 022.
62. Wang, S.; Chen, M.; Huang, D.; Guo, X.; Wang, C. Dream Effected Particle Swarm Optimization Algorithm. *J. Inf. Comput. Sci.* **2014**, *11*, 5631. [[CrossRef](#)]
63. Wu, Z. Optimization of distribution route selection based on particle swarm algorithm. *Int. J. Simul. Model.* **2014**, *13*, 230–242. [[CrossRef](#)]
64. Capel, M.I.; Salguero-Hidalgo, A.; Holgado-Terriza, J.A. Parallel PSO for Efficient Neural Network Training Using GPGPU and Apache Spark in Edge Computing Sets. *Algorithms* **2024**, *17*, 378. [[CrossRef](#)]
65. Nour-El Aine, Y.; Leghris, C. Secure IoT Seed-based Matrix Key Generator. *Int. J. Adv. Comput. Sci. Appl.* **2024**, *15*, 1077–1086. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.