

## CULTURAL INHERITANCE THROUGH SWARM INTELLIGENCE LEARNING

NAILA AZIZA HOUACINE\* AND HABIBA DRIAS

**Abstract.** Cultural inheritance is the main factor for the evolution and skills development through generations. This paper exploits two Machine Learning techniques to imitate the learning process that constitutes the cultural inheritance of Elephants in a Swarm Intelligence (SI) algorithm called the Robotic-Elephant Herding Optimization (REHO). The first REHO-based learning utilizes a GPU k-Nearest Neighbors (GPU-kNN) algorithm, while the second is based on a Deep Neural Network. The performances of the Machine Learning (ML) algorithms are evaluated through five well-known metrics. Their effectiveness in solving the Target Detection Problem is compared with the original REHO via several experiments. The results show that the Deep Learning (DL) model performs better than the GPU-kNN method. These findings prompted us to propose a new hybrid SI and DL model algorithm, namely Robotic-Elephant and Particle Cultural Algorithm (REPCA). This novel approach combines the strength of the REHO-based learning model with the dynamic velocity accelerator of Particle Swarm Optimization (PSO) and knowledge inheritance provided by the Cultural Algorithm. The application problem consists of a swarm of robots that cooperate to detect an exponentially increasing number of targets, known as the Dynamic Target Detection Problem (DTDP). The proposed REPCA is compared with up-to-date algorithms on recent and real-world applications covering the robotics field and the Covid-19 pandemic. The results show a significant positive impact that the cultural inheritance and hybrid methods bring to solve the DTDP.

**Mathematics Subject Classification.** 68T20, 68T05.

Received December 12, 2024. Accepted February 6, 2026.

### 1. INTRODUCTION

Humans and animals are born with genetic materials from their parents. This material is permanent throughout their lives and significantly impacts their becoming. However, it is not the only influential factor. In fact, cultural inheritance also plays an essential role [1]. Cultural inheritance is defined as an inheritance system characterized by the storage and transmission of information by communication, imitation, teaching, and learning. This transmission can take place over a considerable distance and time and between members of a group, clan, community, etc [2].

Cultural inheritance must result from social learning, where the learned notions must be suitable for generalization to similar situations. This concept is well-known in the AI field and is called Machine Learning.

---

*Keywords.* Culture inheritance, swarm intelligence based learning, swarm robotics, deep learning, machine learning, graphics processing unit computation.

LRIA, USTHB, BP 32 El-Alia, Bab-Ezzouar, 16111 Algiers, Algeria.

\*Corresponding author: [naila.houacine@usthb.edu.dz](mailto:naila.houacine@usthb.edu.dz)

Machine Learning allows a machine or a robot to learn from observations. It learns from a set of data, concepts, and knowledge without following explicit instructions. One of the most recent and popular supervised Machine Learning (ML) techniques is Deep Learning. DL technology lies on a structured artificial neural network with numerous layers through which data is transformed to reach a high level of abstraction. It necessitates a considerable amount of data for the training process to learn [3, 4].

In addition, cultural inheritance is widely present in the animal kingdom, especially birds and mammals. It allows behavioral evolution by taking advantage of previous generations' experience to guide and enhance the next generation's behavior and skills [2], which leads us to Swarm Intelligence, another sub-field of AI, that uses nature-inspired optimization algorithms. Their main inspiration is the cooperative behavior of social animals within structured communities that work together at food and water foraging tasks [5]. The main assets of Swarm Intelligence-based algorithms are their flexibility and versatility, their self-learning capability and adaptability to external variations, and their efficiency in solving nonlinear design problems with recent and real-world applications [6].

These advantages are responsible for the great interest and popularity SI algorithms gained over the last decades. In fact, SI algorithms have allowed some improvements in so many fields, such as in electrochemical [7], power networks control and management [8], image processing [9], automation and electrical systems [10], Emergency transportation [11, 12], etc.

Swarm Intelligence techniques have merged with autonomous robots to design what is called Swarm Robotics (SR). The intelligence that characterizes the swarm of autonomous robots makes them more competent and robust to deal with real-world inconsistencies and obstacles. SRs are multi-robot systems where each robot is a simple mobile agent with basic individual abilities, but their strength lies in their union as a swarm. Sharing their individual knowledge with other swarm members reinforces the swarm's global intelligence and builds a better comprehension of their environment, providing more accurate results. One of the benchmark problems of SRs is the Target Detection Problem (TDP), where robots cooperate in the search/detection of one or multiple targets in complex and unknown environments by removing the need for human intervention with reduced cost. The most recent and challenging modeling of the Target Detection Problem is its dynamic version [13] that models the exponentially increasing number of infected COVID-19 persons as our targets.

We proposed two main contributions in this paper. The first consists of the cultural inheritance system implemented through two Machine Learning algorithms that learn the swarm robotics approach: Robotic-Elephant Herding Optimization (Robotic-EHO or REHO). The two REHO-based learning are founded on two different Machine Learning techniques accelerated by GPU: one uses the well-known  $k$ -Nearest Neighbors (kNN) algorithm, while the other utilizes a Deep Neural Network (DNN). The second contribution consists of exploiting the best REHO-based learning to propose a Hybrid Machine Learning – Swarm Intelligence (ML-SI) method for the Dynamic TDP. The advantages of Particle Swarm Optimization (PSO) and the Robotic-EHO cultural inheritance have been combined and provided with an additional cultural dimension to enhance inter- and intra-group communications. The algorithm resulting from this hybridization bears the name of REPCA: Robotic Elephant & Particle Cultural Algorithm. Its performance, efficiency, and stability have been tested against various environments and compared to recent approaches.

The study's originality relies on integrating the cultural dimension into the robot's behavior. To the best of our knowledge, it is the first time that Deep Learning is involved to bring cultural legacy to a Swarm Intelligence algorithm and improve its efficiency. The tools used are from recent emerging technologies such as GPU computation and Deep Learning, and it covers recent and real-world applications such as robotics and the Covid-19 pandemic.

The remainder of the paper is organized as follows. The next section summarizes the recent research related to the TDP. In Section 3, the problem definition and hypothesis are presented. Section 4 describes the comparative study of the kNN and DNN learning of REHO followed by the hybridized Swarm Intelligence learning approach. In Section 5, the experimental results are presented and analyzed. Finally, Section 6 concludes the paper.

## NOTATION

Symbol	Description
$d$	A given day
$z$	A given zone
$\#Target_d$	Number of targets on day $d$
$R_0$	Initial reproduction rate
$R_{d,z}$	Reproduction rate in zone $z$ on day $d$
$CR_{d,z}$	Containment Rate in zone $z$ on day $d$
$\#Elephants$	Number of robotic elephants
$\#clans$	Number of clans
$\alpha$	Influence rate of the best robots position on the clan
$\beta$	Influence rate of the clans gravity center on the best robot position
$i$	Robot's identifier
$j$	Clan's identifier
$X_{i,j}^t, Y_{i,j}^t$	Position of $i$ th robot of the $j$ th clan at time $t$
$X_{Best,j}^t, Y_{Best,j}^t$	Position of the best robot in clan $j$ at time $t$
$X_{Worst,j}^t, Y_{Worst,j}^t$	Position of the worst robot in clan $j$ at time $t$
$size$	The environment side length
$Dist_{i,target}$	Distance between a target and the $i$ th robot's position
$diagSize$	Length of the environments grid diagonal
$F(X_{i,j}^t, Y_{i,j}^t)$	Fitness function of the $i$ th robot, $j$ th clan
$MaxGen$	Maximum number of generation of the algorithm
$MaxTargets$	Maximum number of targets to handle
$MaxVelocity$	Maximum velocity of a robot
$W^t$	Inertia Weigh at time $t$
$W_j^t$	Inertia Weigh of $j$ th clan at time $t$
$X_{GC,j}^t, Y_{GC,j}^t$	Gravity center of the $j$ th clan
$X_{max}, Y_{max}$	Upper bound position that a robot can take
$X_{min}, Y_{min}$	Lower bound position that a robot can take
$MeanF^{t+1}$	Mean fitness function of clan $j$
$Vx_{i,j}^t, Vy_{i,j}^t$	Velocity of the $i$ th robot of the $j$ th clan at time $t$

## 2. RELATED WORK

Numerous are the multi-target search strategies in swarm robotics, such as Swarm Intelligence (SI) based algorithms [14–17], based on the artificial potential field [18, 19], formation-based search strategies [20–22], learning based approaches, and other probabilistic search approaches. However, we may distinguish two main categories among the most recent ones. First, the Swarm Intelligence algorithms, such as:

**GSO:** Glowworm Swarm Optimization [23] is a multi-robot system inspired by the behavior of glowworms that addresses the problem of multiple signal source localization. But studies like [24, 25] have shown after many experiments on multiple benchmarks that GSO has been less efficient and more prone to failure than other algorithms.

**IGES:** The Improved Group Explosion Strategy [26] is dynamically divided into multiple groups, with the ability to split a group if its size is larger than a threshold. In the IGES strategy, each group aims to move its geometric center to the position of its best individual, where the best individual can be selected by four different strategies depending on the size of the group and their previous and current Fitness. These four behaviors allow both inter-group and intra-group cooperation and avoid falling into local optima.

**A-RPSO:** the Adaptive Robotic Particle Swarm Optimization [15] is an improved version of the original PSO. Initially designed for the single-target search issue, the A-RPSO provides an obstacle avoidance strategy

included in the update velocity equation. It also provides an adaptive (dynamic update) of each particle's inertia weight, allowing it to avoid the local optimum.

**MFPSO:** Multi-swarm hybrid of FruitFly Optimization Algorithm (FOA) and Particle Swarm Optimization (PSO) [27] aims at finding a single target in unknown environments. This metaheuristic avoids premature convergence by the independence of the swarms of robots and the evasion of obstacles through the Multi-Scale Cooperative Mutation (MSCM) as an evacuation mechanism from stagnation. Also, an FOA search improves each robot's position, which accelerates the target search.

**E2RPSO:** Exploration-Enhanced Robotic Particle Swarm Optimization [16] is an amelioration of the A-RPSO approach where the swarm's exploration capability is improved to extend to the multi-target search scenarios. The adaptive dynamic inertia weight of A-RPSO is retained while the additional term in the original RPSO method initially used as an obstacle avoidance strategy is modified to enhance the exploration ability by minimizing the revisits of certain areas. The presented results in [16] show that E2RPSO performs better than other PSO-based multi-target search approaches.

From the presented state-of-the-art, we may distinguish that E2RPSO and MFPSO constitute the two most efficient approaches for the studied problem. Thus, we will adapt them to the dynamic TDP for experimental comparison.

The second most eligible category for the multi-target search issue is the formation-based search strategies like the Triangle Formation Search [20], the Probabilistic Finite State Machine-based Strategy (PFSMS) [21] and its Deep Learning-based imitation [22].

**TFS:** In the Triangle Formation Search strategy [20], every swarm of robots is divided into three-robot teams. Each team forms an equilateral triangle, including a leader who determines the moving direction and two members who follow the leader and maintain the formation.

**PFSMS:** The Probabilistic Finite State Machine-based Strategy [21] contains three possible states: diffusion, search, and target processing. In this state machine, each robot at each iteration is in a specific state and has a probability of transferring to other states or maintaining the current one.

**RNSE:** The ReLU No Stacked Auto-Encoders Evolution [22] approach focuses on strategy imitation using a two-stage imitation learning framework. First, they used a deep neural network to learn the behaviors of the PFSMS approach. Then, the parameters of the obtained network are optimized using an evolutionary algorithm. It is considered the first and only application of Deep Learning to swarm robotics for the target search problem.

Since the performance of PFSMS is much better than other presented formation-based search strategies, only PFSMS will be kept from this category for comparison.

We cannot discuss robotic learning tasks without mentioning the significant role of deep reinforcement learning (DRL), which has achieved remarkable success in areas such as autonomous navigation, manipulation, and decision-making [28]. DRL methods excel in learning complex behaviors through interaction with the environment, often surpassing traditional control algorithms in performance for well-defined tasks. In this context, The work in [29] proposes a DRL-based method combining particle filtering, DBSCAN, and deep Q-networks to efficiently localize hazardous sources. In [30], the authors introduced a DRL-based framework that uses uncertainty to guide multi-UAV target search. While other works like [31], and [32] only focus on DRL based robot path planning task.

Despite their strengths, DRL compared to SI-based approaches present several limitations, particularly in the context of robotic target search in unknown environments. They are typically sample-inefficient, requiring large amounts of training data and numerous environment interactions [33]. Furthermore, DRL models are sensitive to hyperparameter settings and reward design, prone to training instability, and can struggle to generalize across environments [34].

### 3. PROBLEM FORMULATION

#### 3.1. Dynamic target detection problem

The modified Target Detection Problem is described in [13] as a dynamic environment simulating COVID-19 spread where the number of cases is exponentially growing. We consider searching multiple static targets with a target versus robot ratio  $>1$  in complex and unknown environments. The environment is defined as a 2D grid-based representation composed of many zones with different Containment Rates (CR) in  $[0, 1]$ . To be as realistic as possible, the Containment Rate is not fixed for a given zone, it changes over the days. CR can increase, decrease, or randomly fluctuate. This CR evolution is calculated by updating its interval boundaries by adding or subtracting 5% of the containment's range, as empirically fixed in [13] to mimic a realistic containment rate variation through days.

Each robot and target occupies a single position  $(X, Y)$ , while the obstacles are rectangular shapes covering multiple grouped positions. Each target emits a perceptible signal whose intensity is inversely proportional to the sensing distance. The total number of targets  $\#Target_d$  on day “ $d$ ” is defined as the sum of all targets across the various zones within the environment. In each zone “ $z$ ”, the number of targets evolves independently according to the zone's reproduction rate  $R_{d,z}$ , applied to the number of targets present in that zone on day “ $d-1$ ”, denoted as  $\#Target_{d-1,z}$ . The number of targets increases through iterations according to equation (1).

$$\#Target_d = \sum_{z=1}^Z R_{d,z} * \#Target_{d-1,z}. \quad (1)$$

$R_{d,z}$  is the reproduction rate of the virus on day “ $d$ ” in zone “ $z$ ”, and its evolution depends on the CR of the concerned zone “ $z$ ”. It is computed using equation (2).

$$R_{d,z} = R_0 * (1 - CR_{d,z}). \quad (2)$$

The initial reproduction rate  $R_0$  represents the reproduction rate of the virus without containment.

#### 3.2. Swarm robotics hypothesis

Swarm Robotics is made up of a group of homogeneous mobile robots. These robots are characterized by a simple computational capacity, a reduced memory, limited autonomy and velocity, and must be equipped with an obstacle avoidance strategy. The robot's visibility range is set up to 10 squares. Within this radius of 10 squares, the robots can perceive/detect the obstacles and the intensity of the signals emitted by the targets.

We assume that each robot can locate its position in the environment, update, and share it with other robots according to the search strategy. However, a robot's motion can be defined in two different ways. Figure 1 exhibits these two techniques, such as on the left side is represented the most commonly used way. It is based on the velocity of each dimension (ex:  $(V_x, V_y)$  in 2D environments or  $(V_x, V_y, V_z)$  in 3D environments). While on the right side, the second manner uses a direction  $\theta$  and distance  $D$  to calculate the next robot's shifting.

### 4. HYBRID APPROACH: REPCA

We propose a hybrid algorithm called REPCA, which is built in four stages illustrated in Figure 2.

**Stage 1:** We develop a machine learning model capable of estimating the direction each robot should take during its target search, inspired by the REHO algorithm. To achieve this, we tested and compared two models: one based on GPU-kNN and the other on DNNs. The best-performing model at this stage is selected as the REHOI model for direction estimation.

**Stage 2:** We leverage the PSO algorithm to estimate the distance each robot needs to travel.

**Stage 3:** We integrate the cultural algorithm to incorporate cognitive aspects that the robots can utilize.

**Stage 4:** Finally, we combine these components into a single hybrid algorithm, REPCA, which synergizes the strengths of the three algorithms to achieve optimal performance.

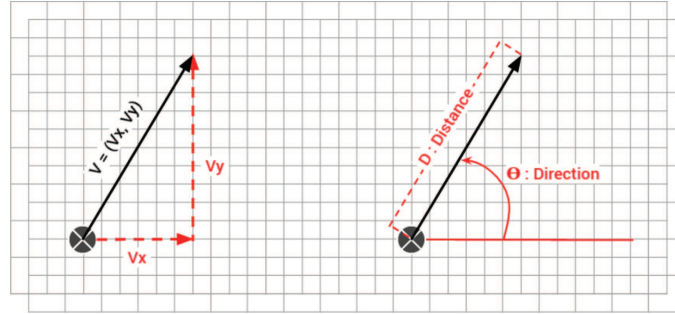


FIGURE 1. Definition of the robot's motion.

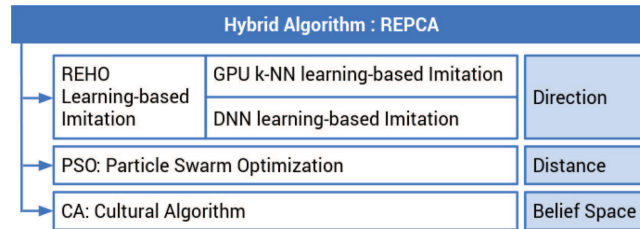


FIGURE 2. General framework of the proposed solution.

### 4.1. REHO learning-based imitation for cultural inheritance

The goal at this stage is to select the most effective REHO learning-based imitation, either the kNN-based or the DNN-based approach, to proceed with the hybridization process. By evaluating the performance of both models, we aim to identify the one that demonstrates superior accuracy and efficiency in estimating the robots' directional decisions. Only the best-performing model will be retained for integration into the hybrid REPCA algorithm, ensuring optimal overall performance.

#### 4.1.1. Original REHO

Robotics Elephant Herding Optimization (REHO) [13] is a Swarm Robotics version of the EHO [35]. REHO came with additional characteristics where robots are equipped with an obstacle avoidance and path planning strategy known as the Discrete Input Space Sampling-based method (DISS) [36]. It also has a limited range of perception and a maximum velocity limit.

Both EHO and REHO are defined by a number of clans “#clans”. Each clan has a number of robot elephants “#Elephants”, the influence rate of the Best robot's position on the rest of the clan “ $\alpha$ ”, and the influence rate of the clan's gravity center on the robot's best position “ $\beta$ ”. The robots' positions update follows the equations below.

The  $i$ th robot of the  $j$ th clan calculates its next position  $(X_{i,j}^{t+1}, Y_{i,j}^{t+1})$  within equation (3).

$$\begin{cases} X_{i,j}^{t+1} = X_{i,j}^t + \alpha * (X_{Best,j}^t - X_{i,j}^t) * r \\ Y_{i,j}^{t+1} = Y_{i,j}^t + \alpha * (Y_{Best,j}^t - Y_{i,j}^t) * r. \end{cases} \tag{3}$$

The fittest robot of each clan estimates its future position  $(X_{\text{Best},j}^{t+1}, Y_{\text{Best},j}^{t+1})$  with equation (4).

$$\begin{cases} X_{\text{Best},j}^{t+1} = \beta * \frac{1}{\#Elephants} * \sum_{i=1}^{\#Elephants} X_{i,j}^t \\ Y_{\text{Best},j}^{t+1} = \beta * \frac{1}{\#Elephants} * \sum_{i=1}^{\#Elephants} Y_{i,j}^t. \end{cases} \quad (4)$$

On the other hand, the robot with the worst fitness in each clan calculates its next position using equation (5). Such as, in our problem modeling, the lower and upper bounds of a position are respectively  $(0, 0)$  and  $(\text{size}-1, \text{size}-1)$ , where “size” is the environment side length and  $\text{rand} \in [0, 1]$ .

$$\begin{cases} X_{\text{Worst},j}^{t+1} = (\text{size}-1) * \text{rand} \\ Y_{\text{Worst},j}^{t+1} = (\text{size}-1) * \text{rand}. \end{cases} \quad (5)$$

The fitness function used to evaluate the robot solutions is defined in equation (6). It depends on three main measurements, *diagSize* which is the length of the environment’s grid diagonal, the Containment Rate  $\text{CR}_{d,z} \in [0, 1]$  of the zone where the robot is located, and the euclidean distance between a target and the current robot position  $\text{Dist}_{i,\text{target}}$ . The greater the fitness value, the closer the robot’s target is.

$$F(X_i, Y_i) = 1 - \text{Min} \left( \frac{\text{CR}_{d,z} + 2 * \frac{\text{Dist}_{i,\text{target}}}{\text{diagSize}}}{3}, \frac{\text{Dist}_{i,\text{target}}}{\text{diagSize}} \right) \in [0 - 1]. \quad (6)$$

These robots cooperate in the search space (environment) on the targets’ search task following Algorithm 1. The algorithm’s objective is to find the positions of all the targets and return a short, safe, and satisfying path from every target’s position to the start position of the clan.

In order to imitate the REHO approach behavior, we need to extract its important features to construct a dataset.

#### 4.1.2. Dataset description

**Inputs:** The data considered relevant as inputs for the learning-based REHO imitation is the one that can impact the calculation of the next position of a robot at an instant t. For this aim, we have identified 18 inputs which are described as: The size of the search environment, the robot’s actual position and its fitness, the clan’s best position and its fitness, the clan’s worst position and its fitness, the number of clans and the number of elephants (robots) in each clan, the influence rate of the best clan’s solution on other robots, the influence rate of the clan’s gravity center on the clan’s best robot, the gravity center position of the clan and its fitness, and a Random number in  $[0, 1]$ .

**Output:** As explained in the previous Section, the calculation of the next position of a robot from its current position can be defined in two ways: The first and most common is based on the velocity of each dimension, therefore  $(V_x, V_y)$  in 2D environments, whereas, the second way uses a direction  $\theta$  (Rad) and a distance  $D$ . The  $V_x$  and  $V_y$  components depend significantly on the environment’s limits. Hence, it seems more judicious to opt for the second strategy that determines the direction that each robot must take, such as: Direction  $\in [-\pi, \pi] = [-3.14159, 3.14159]$ .

**Dataset:** Table 1 summarizes the obtained records for each robot at each iteration after performing more than 40,000 executions of REHO with different combinations of its empirical parameters. Moreover, we have been varying the number of targets in 1, 5, 10, 25, 50 and the size of the complex and unknown environments in 500, 1500, 2500, 3500, 5000. The final dataset is made of 812041 samples.

---

**Algorithm 1.** Original REHO for the dynamic TDP [13].

---

**Input:**  $MaxGen$ ,  $\#Clans$ ,  $\#Elephants$ ,  $\alpha$ ,  $\beta$ ,  $MaxTargets$ ,  $MaxVelocity$ ;  
**Output:**  $PathList$ : List of paths from each target to robot's start position;  
**Begin**  
**Initialization**  
Initialize the generations counter  $t \leftarrow 0$ ;  
Initialize the targets counter  $n \leftarrow 0$ ;  
Initialize the Clan and robot positions  $(x_{i,j}^0, y_{i,j}^0)$ ;  
**while** stopping condition(s) not true **do**  
  **for each** Clan  $j$  **do**  
    Sort all the elephants according to their fitness;  
    **for each** Elephant  $i$  **do**  
      Update position using equations (3)–(5);  
      AdjustVelocity( $MaxVelocity$ );  
      PathPlanning from  $(X_{i,j}^t, Y_{i,j}^t)$  to  $(X_{i,j}^{t+1}, Y_{i,j}^{t+1})$ ;  
      Evaluate the new positions  $(X_{i,j}^{t+1}, Y_{i,j}^{t+1})$  using (6);  
      **if** a target is found **then**  
        Increment the targets counter:  $n + +$ ;  
         $PathList \leftarrow PathList \cup \text{getPath}((X_{i,j}^0, Y_{i,j}^0), (X_{i,j}^{t+1}, Y_{i,j}^{t+1}))$ ;  
      **end if**  
    **end for**  
  **end for**  
  Increment the iterations counter:  $t + +$ ;  
  Update the Environment;  
**end while**  
Return  $PathList$ ;  
**End.**

---

TABLE 1. Statistics about the training dataset.

I/O	Count	Mean	Std	Min	Q1 25%	Median	Q2 75%	Max
size	812 041	2584.29	1474.92	500	1500	2500	3500	5000
$X$	812 041	1252.75	1085.53	0.00	359	952	1913	4999
$Y$	812 041	1260.91	1080.88	0.00	364	976	1934	4999
$F_i$	812 041	0.891	0.132	0.089	0.846	0.935	0.991	1.00
$X_{Best}$	812 041	1208.66	1084.96	0.00	319	892	1869	4997
$Y_{Best}$	812 041	1227.35	1085.65	0.00	331	906	1937	4998
$F_{Best}$	812 041	0.939	0.110	0.115	0.924	1.00	1.00	1.00
$X_{Worst}$	812 041	1224.40	1078.63	0.00	350	917	1843	4999
$Y_{Worst}$	812 041	1218.74	1068.84	0.00	355	918	1830	4999
$F_{Worst}$	812 041	0.817	0.147	0.063	0.751	0.863	0.926	1.00
$\#Clans$	812 041	5.929	0.867	4.00	5.00	6.00	7.00	10
$\#Elephants$	812 041	5.898	0.889	4.00	5.00	6.00	7.00	10
$\alpha$	812 041	0.489	0.140	0.30	0.40	0.50	0.60	0.8
$\beta$	812 041	0.499	0.140	0.30	0.40	0.50	0.60	0.8
$X_{GC}$	812 041	1247.47	1025.61	3.00	403	968	1834	4995
$Y_{GC}$	812 041	1253.21	1020.55	3.00	410	986	1841	4986
$F_{GC}$	812 041	0.790	0.146	0.087	0.708	0.822	0.906	1.00
$r$	812 041	0.393	0.328	0.00	0.046	0.365	0.683	0.999
Direction	812 041	-0.395	1.875	-3.141	-2.114	-0.423	1.155	3.141

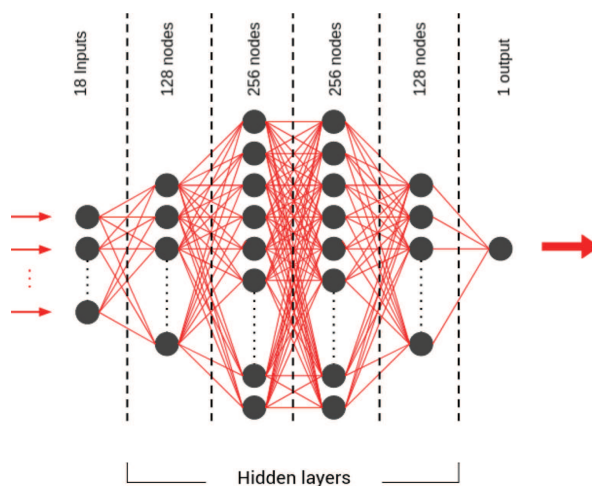


FIGURE 3. Architecture of the DNN.

#### 4.1.3. Deep neural network REHO learning-based Imitation (REHOI-DL)

Jie Li [22] was the first to focus on strategy imitation rather than design by exploiting Deep Learning technologies for the Target Detection Problem. In the same optic, in this section, we propose a DL-based Strategy to learn by imitation of the REHO Algorithm.

In what follows are presented the most important parameters and the structure of our Deep Neural Network, such as the network's architecture and type, the activation function, the weight initialization technique, the optimization algorithm, the learning rate, the batch size, the number of layers, etc.

**Deep neural network properties:** Standardization of Input is the most suitable feature scaling method for Deep Learning. It re-scales the data to a 0-mean and a standard deviation of 1.

It is known that depending on the problem to tackle and the dataset input's type, we may need different neural network structures. Basically, we find that Convolutional Neural Networks (CNN) are the most suitable for image processing. Recurrent Neural Networks (RNN) are further appropriate for sequence processing like Natural Language Processing. When the issue to tackle cannot be associated with any of the two previously cited NN types, we opt for fully-connected Feed-forward Neural Networks (FNN) [22].

To determine the adequate NN architecture, we carried out some tests by varying the number of layers and the number of neurons per layer. The architecture of our fully-connected FNN is composed of four hidden layers with respectively 128, 256, 256, and 128 units as shown in Figure 3. For the optimization algorithms, we found that in recent DL research, mini-batch-based optimizers such as AdaGrad, AdaDelta, RMSProp, Adam, and AdaMax are preferred. The main reasons are that they accelerate the training of the neural networks and compute an unbiased estimation of the expected gradient [37]. After some simple tests, we decided to consider the Adam optimizer, which combines the advantages of AdaGrad and RMSProp. Also, we associated it with a batch size equal to 1000.

The activation function and weight's initializer are two of the most impacting hyper-parameters on the NN performances. Thus, many tests are conducted to combine different options for each parameter to find the best model. *Rectified Linear Unit* (ReLU) and *Scaled exponential Linear Unit* (SeLU) are among the most recent and suitable activation functions for our particular problem [38]. Many weight initializers apply to our problem. We selected a set of three main initializers used in the most recent DL models. There is LeCun [39] that produces weights that are randomly chosen numbers multiplied with the variance  $1/\text{fan-in}$ . It is the most recommended weight initializer when using the SeLU activation function. In the Glorot [40] method, each randomly generated weight is multiplied by variance  $2/(\text{fan-in} + \text{fan-out})$ . The He [41] initializer takes randomly generated weights

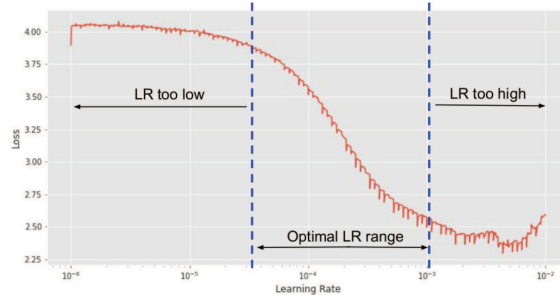


FIGURE 4. Definition of the base and max learning rate values.

TABLE 2. Comparing six types of deep neural networks within five metrics.

Metrics	Init weights	LeCun		He		Glorot	
	Activation f	ReLU	SeLU	ReLU	SeLU	ReLU	SeLU
MAE [Rad <sup>2</sup> ]	Train Loss	<b>0.017 ± 0.001</b>	0.034 ± 0.0005	0.021 ± 0.0009	0.034 ± 0.0009	<b>0.017 ± 0.0007</b>	0.038 ± 0.0009
	Val Loss	0.053 ± 0.003	<b>0.047 ± 0.001</b>	0.057 ± 0.002	0.049 ± 0.001	<b>0.047 ± 0.002</b>	<b>0.047 ± 0.002</b>
	Test Loss	0.053 ± 0.003	<b>0.046 ± 0.001</b>	0.057 ± 0.002	0.049 ± 0.001	<b>0.046 ± 0.001</b>	0.048 ± 0.003
MSE [Rad]	Train Loss	<b>0.019 ± 0.002</b>	0.049 ± 0.001	0.023 ± 0.002	0.045 ± 0.002	0.023 ± 0.002	0.058 ± 0.002
	Val Loss	0.149 ± 0.018	0.096 ± 0.005	0.152 ± 0.007	0.097 ± 0.005	0.127 ± 0.010	<b>0.094 ± 0.013</b>
	Test Loss	0.152 ± 0.015	<b>0.091 ± 0.007</b>	0.151 ± 0.007	0.101 ± 0.006	0.128 ± 0.010	0.096 ± 0.017
R-squared		0.956 ± 0.004	<b>0.974 ± 0.002</b>	0.957 ± 0.002	0.971 ± 0.001	0.963 ± 0.002	0.972 ± 0.004
AIC		-1 377 119.338	<b>-1 753 269.449</b>	-1 380 080.444	-1 676 217.001	-1 501 178.984	-1 719 702.452
BIC		-1 376 912.303	<b>-1 753 062.414</b>	-1 379 873.409	-1 676 009.96	-1 500 971.949	-1 719 495.417

and multiplies them by  $2/\text{fan-in}$ . It is recommended for ReLU activations with Bias tensors initialized to zero where The fan-in and the fan-out are the layer's number of inputs and outputs, respectively.

The Learning Rate (LR), commonly described as the most crucial hyper-parameter in DNN training, needs to be very well-tuned. Smith in [42] proposed a new Cyclical Learning Rate (CLR) to exempt the tuning step by periodically varying it between a minimum and a maximum LR. In order to fix reasonable learning rate boundaries, it is necessary to examine the impact of some LR values on our model, as exhibited in Figure 4.

From Figure 4, we can observe a slight decrease in the Loss from a Learning Rate of  $10^{-5}$ , which shows that the model starts improving. The selection of the lower bound of the Cyclical LR is made a little bit after that, *i.e.*,  $\text{base\_lr} = 0.00005$ . Then, the Loss continues to fall until it reaches a plateau between  $10^{-2}$  and  $10^{-3}$ , after which loss values increase again. Hence, the upper bound of the CLR is fixed to  $\text{max\_lr} = 0.001$ .

Three CLR policies are proposed in [42]: Triangular policy, Triangular2 policy, and Exponential range policy ( $\text{exp\_range}$ ). After a few trials, the  $\text{exp\_range}$  CLR has been chosen. In the  $\text{exp\_range}$  CLR, the upper boundary value declines by an exponential factor of  $\gamma^{\text{iteration}}$  with  $\gamma = 0.999995$ .

Two parameters (Weights initializer and Activation function) are selected with 2 and 3 potential values, respectively. We used a grid search that consisted of trying all the combinations. Each network's configuration has been trained 15 times, resulting in 15 models from which we calculated each metric's mean and standard deviation. The obtained results are summarized in Table 2, the best results are marked in **boldface**.

From the observed results of Table 2, we can deduce that the models using the LeCun–SeLU combination (highlighted in grey) achieve the best performances. After that, we trained 10 NNs with this best configuration and saved them.

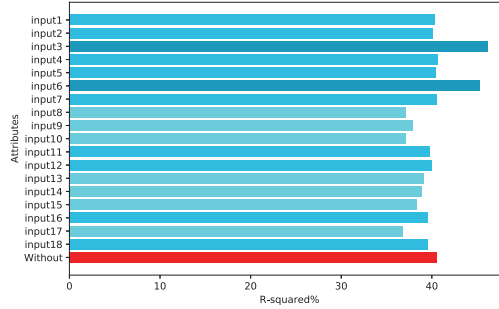


FIGURE 5. Impact of separately weighted inputs on kNN performance.

4.1.4. *k*-Nearest Neighbors REHO learning-based imitation (REHOI-kNN)

The kNN method’s strength consists of its simplicity and low error rate [43]. Nevertheless, it has a non-negligible weakness, such that it is computationally intensive, especially when the size of the train set is huge. However, as mentioned in [4], some solutions have been proposed to overcome this issue. The most interesting one consists of a GPU-based kNN, where parallel computing reduces the running time to a constant  $O(1)$ . This section explores the kNN regression capabilities for the learning strategy imitation of the REHO Algorithm. The very suitability of kNN for the parallel implementation and the results presented in [43] prompted us to opt for a GPU-based approach.

**kNN properties:** Min-max normalization is commonly used to transform all the attribute’s ranges to  $[0, 1]$ . kNN uses distance metric to measure similarity or closeness as a neighbor. Given the nature of the tackled problem, the Euclidean distance is the most suitable metric. The only empirical parameter of kNN is its number of neighbors,  $k$ , and its value depends on the problem to solve. Therefore, we proceeded to the experiment of kNN with different values of  $k$  between 1 and 20. For each  $k$ -value, we made 10 executions shuffling the dataset each time and calculated the mean performances. The obtained results show that the best value of  $k$  is  $k = 13$ .

As mentioned in [44], kNN assumes the equal relevance of all its attributes by normalizing the dataset. However, some features can be more important in the regression (or classification) process. Hence, the kNN method has been modified to incorporate attribute weighting. Such as, a weight has to be assigned to each attribute, depending on its relevance and impact on kNN error [44]. To determine the relevant attributes, it is necessary to calculate the causation between each input’s dimension and the resulting regression performance. Figure 5 shows the obtained results.

The distance calculation formula became as shown in equation (7).

$$d(A1, A2) = \sqrt{\sum_{i=1}^m (\text{weight}_i)^2 * (a1_i - a2_i)^2}. \tag{7}$$

It is good practice for the output calculation to weigh the samples according to the distance that separates them from the test instance. So the resulting output will be closer to the most similar instance among the  $k$ . Equation (8) shows how the output  $B_{test}$  of a test instance  $A_{test}$  is calculated from the weighted mean of its  $k$ -Nearest Neighbors  $\langle A_1, A_2, \dots, A_k \rangle$ .

$$B_{test} = \frac{\sum_{i=1}^k (B_i * (d_{Total} - d(A_{test}, A_i)))}{\sum_{i=1}^k (d_{Total} - d(A_{test}, A_i))} \tag{8}$$

where  $d_{Total}$  is the sum of all the distances between  $A_{test}$  and its  $k$ -Nearest Neighbors.

**kNN algorithm:** As shown in Figure 6, the principle of the GPU-based kNN method is to launch all the distance calculations of the train set in parallel, which means that all the training instances of T are compared

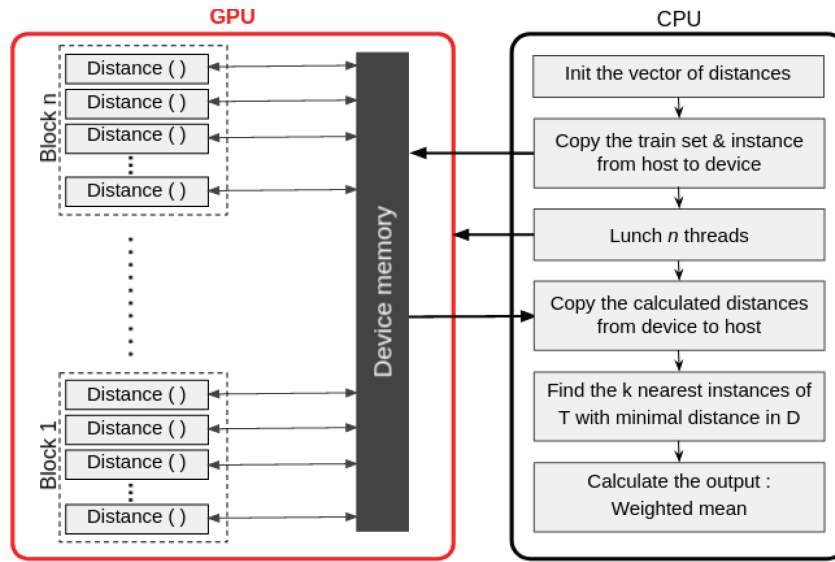


FIGURE 6. The principle of the GPU-based kNN.

to the test instance  $A$  at the same time. The main code (master) is executed on the CPU, and the distance calculation is offloaded to the GPU.

Algorithms 2 and 3 summarize the main steps of the master and kernel, respectively. The GPU-kNN algorithm starts by copying the training set “ $T$ ” and the test instance “ $A$ ” from the host memory to the device memory so the GPU can use them. Then, it launches the required number of threads. Each established thread performs the distance calculation between one instance of the train set and the given test instance.

---

**Algorithm 2.** GPU-kNN (Master).

---

**Input:**  $T$ : Training set;  $A$ : Test instance,  $k$ : number of neighbors;  
**Output:**  $B$ : the output of the test instance;  
**Begin**  
 $n \leftarrow size(T)$ ; ▷ number of instances in the train set.  
 $D$ ; ▷ vector of size  $n$  to store the calculated distances.  
 $CudaMemCpy(T, CudaMemCpyHostToDevice)$ ;  
 $CudaMemCpy(A, CudaMemCpyHostToDevice)$ ;  
 $Calculate\_distance \langle \langle \langle GridDim, BlockDim \rangle \rangle \rangle (T, A, D)$ ; ▷ Launch  $n$  threads  
 $CudaSynchronize()$ ; ▷ synchronize all the threads  
 $CudaMemCpy(D, CudaMemCpyDeviceToHost)$ ;  
 $ListK = FindTheK(T, D, k)$ ; ▷ find the  $k$  instances of  $T$  with minimal distance in  $D$ .  
 $B = Weighted\_mean(ListK, D, k)$ ; ▷ calculate the output using equation (8).  
**Return**  $B$ ;  
**End.**

---

Once all started threads complete their execution, the master recovers the calculated distances in the variable  $D$  and copies them from the Device memory to the Host. After that, it selects the  $k$  instances from  $T$  with the smallest value in the obtained distance vector  $D$ , representing  $A$ ’s  $k$ -Nearest Neighbors. Finally, it calculates the  $weighted\_mean$  of the  $k$  output’s instances to predict  $A$ ’s output.

TABLE 3. Comparison of the two ML performances.

Approaches	MAE	MSE	R-squared	AIC	BIC
DL	<b>0.044537</b>	<b>0.079734</b>	<b>97.73%</b>	<b>-1 642 923.8</b>	<b>-1 642 718.88</b>
kNN	1.462461	0.765496	58.38%	246 977.00	247 193.29

---

**Algorithm 3.** Calculate\_distance (Kernel).

---

**Input:**  $T$ : Train set;  $A$ : Test instance;  $D$ : Distance vector;

**Output:**  $D$ : Distance vector;

**Begin**

$index = blockIdx.x * blockDim.x + threadIdx.x$ ;

$D[index] = \text{Distance}(T[index], A)$ ;

▷ using equation (7).

End.

---

#### 4.1.5. REHOI algorithm

Both the Deep Learning and k-Nearest Neighbors learning-based imitation of REHO can be similarly involved in the Dynamic Target Detection Problem. The REHOI algorithm works the same way as REHO [13], except for the robot's position update method, which is inferred from the inherited behavior resulting from the ML techniques. In the REHOI-DL algorithm, the direction is deduced by the DL model, whereas the REHOI-kNN approach calculates it based on the training set. After that, the new positions are computed using equation (9).

$$\begin{cases} X_{i,j}^{t+1} = X_{i,j}^t + (\cos(\text{direction}) * \text{distance}) \\ Y_{i,j}^{t+1} = Y_{i,j}^t + (\sin(\text{direction}) * \text{distance}) \end{cases} \quad (9)$$

where  $(X_{i,j}^{t+1}, Y_{i,j}^{t+1})$  and  $(X_{i,j}^t, Y_{i,j}^t)$  are the new and current robot's position, respectively. The distance is a random distance in  $[1, \text{MaxVelocity}]$ .

#### 4.1.6. Evaluation and comparison (DL, kNN, REHO)

As explained, this part of the work necessitates using a GPU to accelerate the learning phase of our two learning-based imitation algorithms. We have trained the Deep Learning Model using the *Python* language with *Tensorflow* and *Keras*, while the kNN proposed approach uses *C-CUDA* language on a CPU host coupled with a GPU (*NVIDIA Quadro 5000*) device.

**Machine learning performances:** Five metrics were selected to evaluate and compare the performances of the two machine Learning algorithms (kNN and the DL). These metrics are The Mean Squared Error (MSE), the Mean Absolute Error (MAE), Akaike's Information Criterion (AIC), the Bayesian Information Criterion (BIC), and the coefficient of determination (R-squared) [45].

We ran numerous tests with many parameter variations of both the kNN approach and the DNN model, and the best-obtained results are presented in Table 3, marked in **boldface**. The DL model appears to perform much better than the kNN algorithm in this REHO learning strategy imitation task. It has achieved a much smaller error, AIC, BIC, and a significantly higher percentage variation of the output explained by the given inputs (R-squared).

**REHO vs. REHOI performances:** A series of experiments on the Target Detection Problem was conducted to compare the original REHO with the proposed REHOI-DL and REHOI-KNN. Each approach has been executed 20 times in 5 different medium complex environments ( $2500 \times 2500$ ) with an increasing, random, and diminishing dynamic containment rate (with  $R_0 = 2.5$ ). The initial target's number is fixed at 100, the total number of robots equals 50, and their speed is valued at 15 iterations/day. Table 4 shows the mean  $\pm$  std itera-

TABLE 4. Average performances of the compared approaches.

Approaches	REHO	REHOI-DL	REHOI-kNN
Mean Iterations	17.74 ± 31.9	<b>52.04 ± 96.3</b>	55.19 ± 104.6
Mean Exe time	55.80 ± 422.8	<b>164.05 ± 522.7</b>	165.28 ± 417.7
Mean Success rate	0.9906	<b>0.8888</b>	0.8885

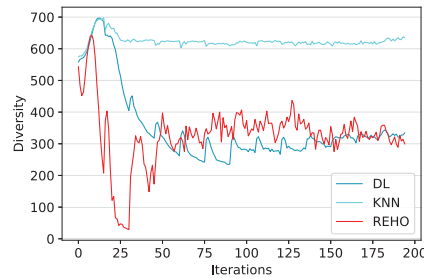


FIGURE 7. Diversity variation of the three approaches.

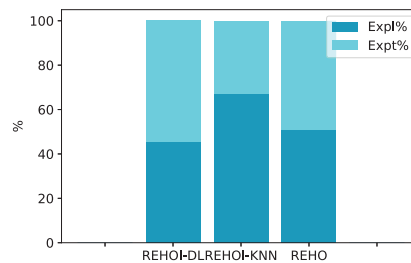


FIGURE 8. Mean exploration and exploitation balance.

tions, execution time (in seconds), and the average success rate of each algorithm. The REHOI-DL performance, marked in **boldface**, surpassed that of REHOI-kNN.

Regarding iterations' number, REHOI-DL is closer to REHO performance than REHOI-kNN, whereas the execution time and success rate of both REHOI approaches are similar but less efficient than the original REHO. These results can be explained by studying the three algorithms' behaviors in terms of diversity [46] and exploration (Expl%) *vs.* exploitation (Expt%) balance [47].

As shown in Figure 7, the initial diversity of all three approaches starts at the same point. The diversity variation of REHOI-DL and REHO oscillate until they get close values around 300, whereas REHOI-kNN has a stable high diversity estimated at 600. Figure 8 depicts the Exploration *vs.* Exploitation balance of the three compared methods, with 45.28/54.71 for REHOI-DL, 67.19/32.80 for REHOI-kNN, and 50.98/49.01 for the original REHO. REHO and REHOI-DL are the most balanced ones compared to REHOI-kNN, which promotes exploration to the detriment of exploitation, which seems to reduce its performance.

The variation of the exploration and exploitation ratios of the compared approaches during the search process is represented in Figures 9–11. It confirms the analysis of the previous results. REHOI-DL has the most similar behavior to the original REHO. Consequently, REHOI-DL realizes the closest performance to REHO.

Comparing the two learning strategy imitations of the REHO shows that the approach exploiting the Deep Learning technology (REHOI-DL) is the most adapted one for further work on this paper.

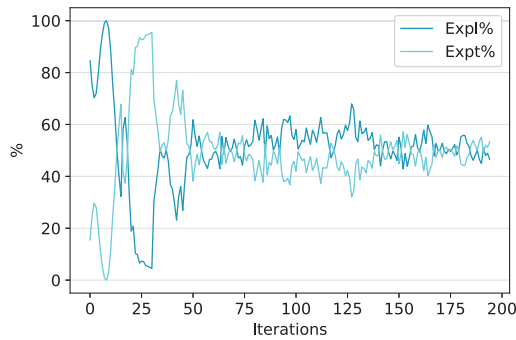


FIGURE 9. REHO's Expl/Expt variation through iterations.

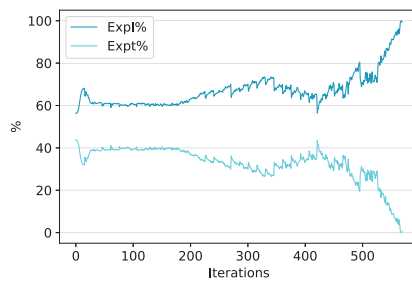


FIGURE 10. REHOI-kNN's Expl/Expt variation through iterations.

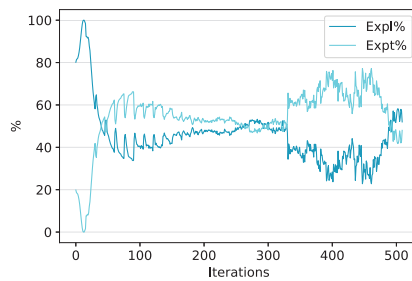


FIGURE 11. REHOI-DL's Expl/Expt variation through iterations.

## 4.2. Original PSO

Particle Swarm Optimization (PSO) is a Swarm Intelligence-based and population-based algorithm. It consists of a swarm of particles that represents the candidate solutions. Each particle is moving in the search space seeking the optimal solution. The particles' movement is mainly guided by their position, velocity, personal best solution, and the best solution among the entire swarm. Since its first introduction in 1995 by Kennedy and Eberhart [48], several versions of PSO have been proposed, and some new parameters have been introduced. One of the most important and impacting parameters of modern PSO is the "Inertia Weight" that we refer to as "W". It has been known that "W" has a crucial role in balancing between exploration and exploitation processes. It regulates the practicals' convergence speed and guarantees the robustness of PSO in consequence. From there, numerous Inertia Weight strategies have been proposed and adapted to each optimization problem.

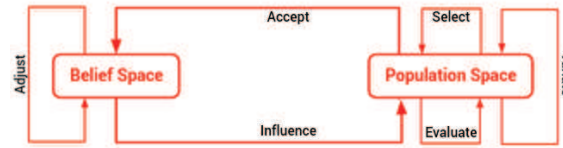


FIGURE 12. Cultural algorithm principle.

Bansal *et al.* [49] and Rathore and Sharma [50] presents an overview of the most recent and popular Inertia Weight strategies.

As mentioned in the related work, PSO-based algorithms have been widely used in robotic target searching and proved to be successful in optimizing the search type problem especially in locating targets in hazardous environments [15, 16, 27]. Accordingly, we decided to integrate this key parameter of PSO and especially to use the exponential Weight inertia [51] shown in equation (10) in the proposed method.

$$w^t = w^{\text{init}} * e^{-a \left( \frac{t}{\text{maxGen}} \right)^b} \quad (10)$$

where  $w^{\text{init}}$  is set to 0.9 and  $a = b = 1.0$  for balanced search between global and local search as recommended in [51].

### 4.3. Original CA

Reynolds first introduced cultural algorithms [52] as a class of computational models representing societies' cultural evolution. As shown in Figure 12, CA consists of three components: (1) a population, which represents some possible solutions to the problem, (2) a belief space that englobes five types of knowledge about the individuals' experience, including situational, normative, historical, topographical, and domain knowledge. (3) A communication protocol as an interface between the population and its beliefs. This protocol includes three functions: Accept, Adjust, and Influence. The accept function determines which individuals of the current population are selected to impact the belief space. The Adjust function can be described by the action of the best individuals of the population in the belief space. And finally, The influence function is responsible for transmitting the belief space's knowledge to the population to guide its evolution.

The Cultural Algorithm has proven that the transmission of the beliefs and knowledge that each generation has gained enables the search strategy to achieve higher results, even with a small population. So, it seems relevant to introduce the beliefs' dimension of the CA to the resolution method in order to get better results with a reasonable population size for the Dynamic TDP.

### 4.4. Combination REPCA

The proposed Robotic Elephant & Particle Culture Algorithm hybridizes a Deep Learning-based imitation of REHO with PSO and CA algorithms. Combining the direction calculation of REHOI and the distance estimation by PSO forms a hybrid optimization method to which the CA brings a cultural dimension. This last one also provides a knowledge legacy between elephants of the same clan and between different clans across generations.

As presented in Algorithm 4, the proposed approach starts with the initialization step. It initializes the population space (Ps) by randomly generating a number of "PopSize" robots grouped in clans of  $\#Elephants$  elephants. It also initializes the Belief space (Bs). While the algorithm has not satisfied the stop conditions, it selects the robots that will update the Belief space. After that, it adjusts the belief space to consider the selected population's knowledge. These last influence the evolution of the next generation of the Population space. The population evolves according to the influence gained from the belief space, so it updates the robots' positions using a combination of REHOI and PSO strategies and evaluates them.

**Selection and acceptance:** The selection of the individuals from the current population that will influence the current beliefs space uses absolute ranking based on the mean fitness value of each clan. Such as, we select the top  $\#Clans$  Clans that maximize the mean fitness function.

**Adjusting the belief space:** The update of the belief space is based on the new population positions of the selected robots. It includes adjusting both the Situational and Normative knowledge, such as updating the best position, best fitness, worst position, worst fitness, Gravity center (GC) position of each clan, and the mean fitness of each clan according to equations (11)–(15), respectively.

$$\left( X_{Best,j}^{t+1}, Y_{Best,j}^{t+1} \right) = \begin{cases} \left( X_{i,j}^t, Y_{i,j}^t \right); & \text{if } F(X_{i,j}^t, Y_{i,j}^t) > F(X_{Best,j}^t, Y_{Best,j}^t) \\ \left( X_{Best,j}^t, Y_{Best,j}^t \right); & \text{otherwise} \end{cases} \quad (11)$$

$$F\left( X_{Best,j}^{t+1}, Y_{Best,j}^{t+1} \right) = \begin{cases} F\left( X_{i,j}^t, Y_{i,j}^t \right); & \text{if } F\left( X_{i,j}^t, Y_{i,j}^t \right) > F\left( X_{Best,j}^t, Y_{Best,j}^t \right) \\ F\left( X_{Best,j}^t, Y_{Best,j}^t \right); & \text{otherwise} \end{cases} \quad (12)$$

$$\left( X_{Worst,j}^{t+1}, Y_{Worst,j}^{t+1} \right) = \begin{cases} \left( X_{i,j}^t, Y_{i,j}^t \right); & \text{if } F\left( X_{i,j}^t, Y_{i,j}^t \right) < F\left( X_{Worst,j}^t, Y_{Worst,j}^t \right) \\ \left( X_{Worst,j}^t, Y_{Worst,j}^t \right); & \text{otherwise} \end{cases} \quad (13)$$

$$F\left( X_{Worst,j}^{t+1}, Y_{Worst,j}^{t+1} \right) = \begin{cases} F\left( X_{i,j}^t, Y_{i,j}^t \right); & \text{if } F\left( X_{i,j}^t, Y_{i,j}^t \right) < F\left( X_{Worst,j}^t, Y_{Worst,j}^t \right) \\ F\left( X_{Worst,j}^t, Y_{Worst,j}^t \right); & \text{otherwise} \end{cases} \quad (14)$$

$$\left( X_{GC,j}^t, Y_{GC,j}^t \right) = \left( \frac{1}{\#Elephants} * \sum X_{i,j}^t, \frac{1}{\#Elephants} * \sum Y_{i,j}^t \right). \quad (15)$$

Using the fitness function of each robot  $i$  of clan  $j$ , the mean fitness function of clan  $j$  can be calculated according to equation (16).

$$MeanF_j^{t+1} = \frac{1}{\#Elephants} * \sum_{i=1}^{\#Elephants} F\left( X_{i,j}^{t+1}, Y_{i,j}^{t+1} \right). \quad (16)$$

**Influence functions:** The belief space guides the new generation by adapting some of its parameters. As shown in equation (17), we update the inertia weight of each clan depending on the knowledge components of the belief space. If a given Clan  $i$  goes in the right direction, the exponential inertia weight formula is used. Otherwise,  $W^t$  is re-initialized with  $W^{init}$  value to boost diversification.

$$W^t = \begin{cases} W^{init} * e^{-a\left(\frac{t}{MaxGen}\right)^b}; & \text{if } MeanF_j^t \leq MeanF_j^{t+1} \\ W^{init}; & \text{otherwise.} \end{cases} \quad (17)$$

**Variate population:** A combination of the Deep Learning-based imitation of REHO (REHOI) and the PSO approach has been proposed for the population variation. This hybrid algorithm considers the belief space's influence to enhance the population's quality. It aims to maximize the coverage of the search space by increasing its diversity. Equation (18) describes the update formula of the worst robot of each clan. It offers more diversification and prevents the clan from being stuck at a local optimum by randomly exploring a new zone.

$$\begin{cases} X_{Worst,j}^{t+1} = (size-1) * rand, & \text{with } rand \in [0, 1] \\ Y_{Worst,j}^{t+1} = (size-1) * rand, & \text{with } rand \in [0, 1]. \end{cases} \quad (18)$$

The rest of the robots calculate their direction thanks to the Deep Learning model obtained in Section 4.1. It takes as input all the 18 information about the actual position and fitness of the robot, the gravity center of its clan, alpha and beta parameters, etc. The velocity of each robot is updated as shown in equation (19), using their old velocity and the estimated exponential inertia weight  $W_j^t$  of the  $j$ th clan. This new velocity is involved in calculating the distance each robot must traverse, according to equation (20). Thus, for each

robot, from its calculated distance and direction, its new position is approximated through equation (21).

$$\begin{cases} Vx_{i,j}^{t+1} = (W_j^t * Vx_{i,j}^t) \% Vmax \\ Vy_{i,j}^{t+1} = (W_j^t * Vy_{i,j}^t) \% Vmax \end{cases} \quad (19)$$

$$\text{distance} = \sqrt{Vx_{i,j}^{t+1}{}^2 + Vy_{i,j}^{t+1}{}^2} \quad (20)$$

$$\begin{cases} X_{i,j}^{t+1} = X_{i,j}^t + (\cos(\text{direction}) * \text{distance}) \\ Y_{i,j}^{t+1} = Y_{i,j}^t + (\sin(\text{direction}) * \text{distance}). \end{cases} \quad (21)$$

Since this is a robot-oriented approach, real-world scenarios must be implemented considering the MRSs characteristics. Thus, in addition to the limited range of perception and a maximum velocity limit,  $Vmax$ , robots must provide obstacle avoidance and path planning strategies [13]. Once the new positions of robots are calculated, the robot navigation method (path planning method) is invoked to build a short and obstacle-free (safe) path from the current robot position  $(X_{i,j}^t, Y_{i,j}^t)$  to the next one  $(X_{i,j}^{t+1}, Y_{i,j}^{t+1})$ .

**Evaluation:** The fitness function used to evaluate the robot solutions is the same used in the original REHO algorithm, defined in equation (6). The greater the fitness value, the closer the target.

**Stop conditions:** To terminate the targets' search mission, there are 3 termination conditions. First, if the maximum number of generations  $MaxGen$  is reached. Second, if the search time limit is exceeded  $MaxTime$ , it aborts the mission. Finally, if all the targets are found, then the mission succeeds.

## 5. EXPERIMENTS AND RESULTS

The validation of the proposed approach goes through an experimental study. A series of comparative tests are conducted under different problem settings. For an accurate and unbiased comparison of the algorithms' performances, all the methods have been implemented in C++ on the same machine. Each confronted approach has been executed on each parameter 30 times in 5 different environments. The mean iterations' number, the mean execution time (seconds), and the mean success rate (mean percentage of found targets among all the targets of the environment) are the considered metrics for the comparison.

In the following part, the parameter settings of the compared methods are presented. REPCA parameters tuning have been conducted through the self-parameterizer [53], where the other approaches are getting their parameters from their original paper. After that, the impact of 3 parameters is studied, the impact of the initial number of targets, the impact of the population size, and the influence of the initial reproduction rate  $R_0$ . Each of these parameters is experienced under three scenarios of Containment Rate evolution: growing Containment Rate, randomly changing Containment Rate, and diminishing Containment Rate. Some settings related to the environment and the robot's capacities are fixed, such as the environment's size =  $2500 \times 2500$  squares, the maximum number of iterations = 1000, the search time limit = 300 s, the maximum velocity set to 100, and the robots speed = 15 iterations per day.

### 5.1. Settings of the compared algorithms

The comparison algorithms and their corresponding parameter configurations are presented as follows:

**E2RPSO:** Robotic Particle Swarm Optimization, initial inertia weight  $wini = 0.9$ ,  $\alpha = 0.5$ ,  $\beta = 0.7$ , the cognitive constant  $C1 = 0.4$ , the social constant  $C2 = 0.9$ , the obstacle avoidance coefficient  $C3\_index = 5$ , and the map area exploration rate increment  $K = 1$ .

**PFSMS:** Probabilistic Finite State Machine-based Strategy, it is composed of 40 teams forming triangle shapes of side length = 20,  $\alpha = 0.75$ , and  $\beta = 1.33$ .

**REHO:** The Robotic-Elephant Herding Optimization with a population size of 40 individuals (10 clans with 4 elephants per clan),  $\alpha = 0.5$ , and  $\beta = 0.6$ .

**Algorithm 4.** REPCA algorithm.**Input:**  $MaxGen$ ,  $\#Targets$ ,  $\#Clans$ ,  $\#Elephants$ ,  $\alpha$ ,  $\beta$ ,  $MaxVelocity$ ,  $a$ ,  $b$ ,  $W^{init}$ ;**Output:**  $PathList$ : All paths from each target to the robot's start position;**Begin****Initialization**Set the generations counter,  $t \leftarrow 0$ ;Initialize: population  $Ps^0$  and belief space  $Bs^0$ ;Load the DL model:  $DLModel$ ;**while** stopping condition(s) not true **do**    Select  $\#Clans$  Clans of robots from  $Ps^t$ ;    Adjust the  $Bs^t$  using equations (11)  $\rightarrow$  (16);    Influence the  $Ps^t$  using equation (17);    Variate the  $Ps^t$ ;    Compute each robot's Direction using the  $DLModel$ ;

Calculate each robot's Distance using equations (19) and (20);

Determine the new robot positions using equations (18) and (21);

    AdjustVelocity( $MaxVelocity$ );    Robot Path Planning from  $(x_{i,j}^t, y_{i,j}^t)$  to  $(x_{i,j}^{t+1}, y_{i,j}^{t+1})$ ;

Evaluate the fitness of each robot using equation (6);

**if** a target is found **then**        increment the targets counter:  $n++$ ;        Re-init the clan iterations counter:  $CountT_j \leftarrow 0$ ;         $PathList \leftarrow PathList \cup getPath((x_{i,j}^0, y_{i,j}^0), (x_{i,j}^{t+1}, y_{i,j}^{t+1}))$ ;    **end if**    Increment the clan iterations counter:  $CountT_j++$ ;    Increment the global iterations counter:  $t++$ ;

Update the environment;

**end while**Return  $PathList$ ;**End.**

**REPCA:** The proposed Robotic-Elephant & Particle Cultural Algorithm. Its population size is set to 40 individuals (10 clans with 4 elephants each),  $\alpha = 0.5$ ,  $\beta = 0.6$ , an initial inertia weight fixed at 0.9, and  $a = b = 1.0$ .

## 5.2. Influence of the population size

In this subsection, the impact of the population size on the different search algorithms is experimented. The population size has been varied within  $\langle 20, 30, 40, 50, 60 \rangle$  individuals, the initial reproduction rate " $R_0$ " fixed to 2.5 according to the COVID-19 initial " $R_0$ " in China and several European countries [54, 55], and the initial number of targets " $\#Target_0$ " to 100. Table 5 shows the obtained results in environments with a growing, random, and diminishing CR.

Globally, the results in a Growing CR are better than the ones in a Random CR, and the results in a Diminishing CR are close to or worse than the Random CR. But in all the scenarios, the REPCA algorithm is the only one to find 100% of the targets with 50 or more robots in a very reasonable time and iteration number. Followed by the REHO algorithm, then the PFSMS one, and finally, E2RPSO.

Figures 13–15 exhibit the impact of the population size on the four compared algorithms in a Random Containment Rate. It shows that when increasing the population size (total number of robots), all the algorithms' performance is enhanced in reaching targets. The number of iterations considerably decreases for the PFSMS and REPCA approaches, while it does not vary much for the 2 other algorithms. For the execution time, only REPCA tends to take less time with the population augmentation because of its increasing efficiency. On the other side, the three other methods reach the execution time limit of 300 s, leading to the mission's failure. It is

TABLE 5. Average iterations number, execution time, and success rate with different population sizes.

PopSize	E2RPSO			PFSMS			REHO			REPCA		
	Iter	Time	Succ	Iter	Time	Succ	Iter	Time	Succ	Iter	Time	Succ
Growing CR												
20	131 ± 0.7	301.9 ± 0.8	0 ± 0.0	109 ± 0.7	313.5 ± 7.7	0 ± 0.0	82 ± 7.1	177.3 ± 69.5	3 ± 0.0	114 ± 19.5	309.9 ± 7.5	11 ± 0.0
30	96 ± 0.6	301.9 ± 0.9	0 ± 0.0	106 ± 1.2	312.0 ± 10.0	0 ± 0.0	92 ± 2.4	277.0 ± 40.3	5 ± 0.0	167 ± 125.0	199.6 ± 145.6	48 ± 0.4
40	72 ± 0.4	301.4 ± 0.9	0 ± 0.0	105 ± 1.3	309.8 ± 5.9	1 ± 0.0	97 ± 5.7	292.6 ± 25.8	10 ± 0.0	109 ± 146.2	98.4 ± 138.1	79 ± 0.3
50	58 ± 0.5	302.7 ± 1.4	0 ± 0.0	105 ± 1.7	310.7 ± 8.7	2 ± 0.0	103 ± 7.2	309.7 ± 7.4	14 ± 0.0	16 ± 1.7	8.5 ± 0.5	100 ± 0.0
60	48 ± 0.3	301.9 ± 1.8	0 ± 0.0	106 ± 2.1	313.6 ± 12.1	2 ± 0.0	112 ± 10.5	309.2 ± 6.8	18 ± 0.0	14 ± 1.6	8.9 ± 0.7	100 ± 0.0
Random CR												
20	131 ± 0.5	302.2 ± 1.2	0 ± 0.0	109 ± 0.7	313.8 ± 8.7	0 ± 0.0	82 ± 7.1	177.4 ± 73.3	3 ± 0.0	111 ± 23.1	301.9 ± 54.6	12 ± 0.2
30	95 ± 0.8	301.7 ± 1.2	0 ± 0.0	106 ± 1.2	311.6 ± 10.4	0 ± 0.0	91 ± 2.8	273.1 ± 45.7	5 ± 0.0	134 ± 100.2	180.1 ± 150.3	54 ± 0.4
40	71 ± 0.7	302.2 ± 1.3	0 ± 0.0	105 ± 1.5	310.3 ± 7.9	1 ± 0.0	95 ± 5.0	289.2 ± 27.7	10 ± 0.0	124 ± 121.2	139.0 ± 150.1	69 ± 0.4
50	57 ± 0.5	303.2 ± 1.4	0 ± 0.0	105 ± 1.8	311.5 ± 8.6	2 ± 0.0	101 ± 6.7	309.1 ± 9.9	13 ± 0.0	25 ± 49.0	18.6 ± 55.2	97 ± 0.1
60	48 ± 0.0	304.4 ± 0.3	0 ± 0.0	106 ± 2.2	314.1 ± 9.8	2 ± 0.0	111 ± 15.4	309.9 ± 7.4	18 ± 0.1	14 ± 1.6	8.9 ± 0.5	100 ± 0.0
Diminishing CR												
20	131 ± 0.5	302.0 ± 1.1	0 ± 0.0	109 ± 0.6	316.6 ± 7.1	0 ± 0.0	84 ± 7.4	191.5 ± 76.2	3 ± 0.0	124 ± 37.6	312.1 ± 7.8	10 ± 0.0
30	95 ± 0.0	300.8 ± 0.3	0 ± 0.0	107 ± 0.9	315.7 ± 10.8	0 ± 0.0	91 ± 3.3	275.8 ± 44.9	5 ± 0.0	202 ± 145.3	217.8 ± 137.5	45 ± 0.4
40	72 ± 0.0	302.9 ± 0.1	0 ± 0.0	106 ± 1.8	315.7 ± 10.7	1 ± 0.0	97 ± 5.7	303.4 ± 20.6	9 ± 0.0	77 ± 111.2	79.2 ± 129.3	83 ± 0.3
50	58 ± 0.0	304.8 ± 0.1	0 ± 0.0	106 ± 1.5	312.1 ± 9.0	1 ± 0.0	99 ± 6.1	308.8 ± 7.1	15 ± 0.0	15 ± 1.6	8.2 ± 0.6	100 ± 0.0
60	48 ± 0.0	302.7 ± 0.1	0 ± 0.0	106 ± 1.9	311.9 ± 9.5	2 ± 0.0	108 ± 10.5	311.7 ± 10.5	16 ± 0.1	13 ± 1.3	8.5 ± 0.6	100 ± 0.0

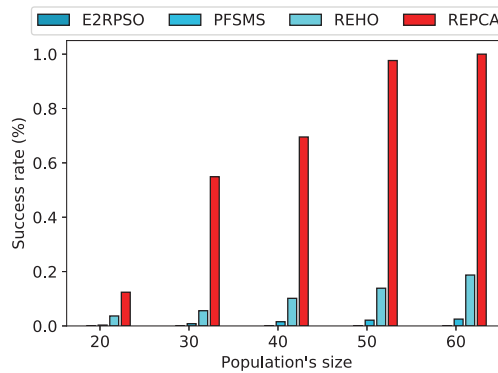


FIGURE 13. Success rate comparison with different population sizes in a random RC.

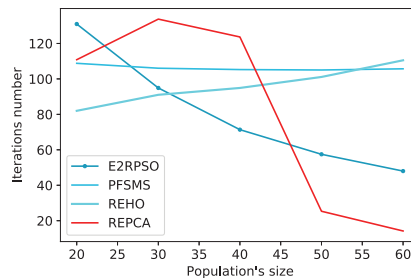


FIGURE 14. Iterations number comparison with different population sizes in a random RC.

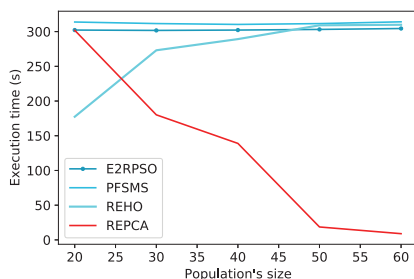


FIGURE 15. Execution time comparison with different population sizes in a random RC.

noticeable that the REPCA approach outperforms comparative algorithms no matter the size of the population. Furthermore, it reached a 100% success rate with a sufficient number of robots, around 60.

These findings show that REPCA is well-suited for real-world applications requiring scalable, fast, and reliable coordination, especially for pandemic response or disaster relief. As the number of robots increases, REPCA not only improves in performance but also reduces execution time, unlike other algorithms that fail to scale. This makes REPCA highly effective in time-critical missions.

In contrast to traditional methods that degrade under increased complexity or agent count, REPCA adapts dynamically, offering a dependable solution for autonomous robot coordination in fast-changing, high-stakes settings.

### 5.3. Influence of the initial number of targets

The impact of the initial number of targets on the different search algorithms is addressed in this part. The initial targets' number has been varied within  $\langle 5, 50, 100, 250, 500 \rangle$  targets, the reproduction rate " $R_0$ " fixed to 2.5, and the population size to 40 individuals. The tests have been conducted in environments with a growing, random, and diminishing CR, and the results are exhibited in Table 6. It shows the ability of REPCA over other methods to efficiently contain the virus spread before it becomes exponential and impossible to manage without taking drastic measures. However, the comparative results of Table 6 reveal that the performance of each algorithm slightly degrades when going from the growing CR mode to the random and diminishing ones.

From the results obtained in the randomly updated CR, we found that as illustrated in Figures 16–18, the success rate diminishes with the growth of the initial number of targets for all the search approaches. This behavior is caused by the additional difficulty of the search. When the number of initial targets is too large, the number of targets increases tenfold at phenomenal speed so that it is difficult to manage or even unmanageable for certain methods. Hence, with the drop in the success rate, the execution time of the compared algorithms increases to reach the maximum time limit of 300 s and ends up aborting the mission earlier, performing fewer iterations.

These findings highlight a key challenge for real-world robotic deployments that is managing a high number of initial targets. In scenarios like disaster response or pandemic containment, when many critical areas are known at launch, all evaluated algorithms show reduced success rates and increased execution time, often leading to aborting the mission, although REPCA still has the best resistance to the increase of this parameter.

This mirrors real-world difficulties in handling large-scale crises from the outset. A high initial workload strains coordination and slows down progress, leading to incomplete coverage. The results underscore the need for scalable, efficient algorithms capable of prioritizing and managing dense target environments right from deployment, ensuring reliable performance under pressure.

TABLE 6. Average iterations number, execution time, and success rate with different initial number of targets.

$Target_0$	E2RPSO			PFSMS			REHO			REPCA		
	Iter	Time	Succ	Iter	Time	Succ	Iter	Time	Succ	Iter	Time	Succ
Growing CR												
5	63 ± 2.8	302.2 ± 1.6	0 ± 0.0	158 ± 55.2	270.1 ± 106.1	14 ± 0.3	30 ± 55.6	30.6 ± 91.2	90 ± 0.3	3 ± 1.1	0.3 ± 0.1	100 ± 0.0
50	63 ± 2.2	302.2 ± 1.2	0 ± 0.0	120 ± 2.5	310.4 ± 8.0	1 ± 0.0	116 ± 9.6	303.8 ± 15.4	11 ± 0.0	12 ± 1.6	3.1 ± 0.3	100 ± 0.0
100	61 ± 3.4	302.7 ± 1.4	0 ± 0.0	105 ± 1.3	311.5 ± 8.4	1 ± 0.0	96 ± 5.9	309.4 ± 14.8	10 ± 0.0	101 ± 135.8	99.5 ± 139.7	77 ± 0.3
250	57 ± 2.1	302.4 ± 1.6	0 ± 0.0	88 ± 0.6	307.5 ± 4.7	2 ± 0.0	75 ± 3.0	303.3 ± 7.9	12 ± 0.0	77 ± 6.7	308.9 ± 7.3	33 ± 0.1
500	57 ± 2.2	302.5 ± 1.7	0 ± 0.0	77 ± 0.5	314.3 ± 10.6	0 ± 0.0	58 ± 2.4	308.4 ± 8.3	13 ± 0.0	40 ± 0.6	309.4 ± 3.8	24 ± 0.0
Random CR												
5	72 ± 0.4	302.4 ± 0.9	0 ± 0.0	157 ± 53.3	269.1 ± 105.7	14 ± 0.3	35 ± 49.0	48.0 ± 107.6	84 ± 0.3	3 ± 0.5	0.3 ± 0.1	100 ± 0.0
50	72 ± 0.5	301.9 ± 1.0	0 ± 0.0	120 ± 1.9	309.8 ± 6.2	1 ± 0.0	116 ± 8.9	298.9 ± 19.7	10 ± 0.0	11 ± 1.2	3.1 ± 0.2	100 ± 0.0
100	71 ± 0.7	302.2 ± 1.3	0 ± 0.0	105 ± 1.5	312.3 ± 11.0	1 ± 0.0	95 ± 4.9	293.3 ± 26.8	9 ± 0.0	71 ± 99.1	77.4 ± 126.2	82 ± 0.3
250	71 ± 0.6	301.8 ± 1.2	0 ± 0.0	88 ± 0.7	309.3 ± 4.8	1 ± 0.0	73 ± 3.0	304.5 ± 11.3	13 ± 0.0	78 ± 5.6	309.1 ± 7.0	29 ± 0.1
500	71 ± 0.7	302.3 ± 1.4	0 ± 0.0	77 ± 0.5	314.1 ± 9.0	0 ± 0.0	59 ± 2.2	310.1 ± 11.4	12 ± 0.0	40 ± 0.9	307.8 ± 5.1	24 ± 0.0
Diminishing CR												
5	72 ± 0.0	302.7 ± 0.2	0 ± 0.0	151 ± 62.5	258.7 ± 115.9	17 ± 0.4	56 ± 71.9	67.0 ± 121.7	78 ± 0.4	3 ± 0.3	0.3 ± 0.0	100 ± 0.0
50	72 ± 0.0	302.7 ± 0.1	0 ± 0.0	120 ± 2.3	311.3 ± 9.3	1 ± 0.0	118 ± 10.0	299.8 ± 22.4	11 ± 0.0	11 ± 1.2	2.9 ± 0.2	100 ± 0.0
100	72 ± 0.0	302.9 ± 0.1	0 ± 0.0	106 ± 1.4	311.9 ± 10.4	1 ± 0.0	94 ± 3.6	295.7 ± 25.1	9 ± 0.0	52 ± 87.8	47.4 ± 100.8	90 ± 0.2
250	72 ± 0.2	303.1 ± 0.4	0 ± 0.0	88 ± 0.7	307.1 ± 4.3	1 ± 0.0	73 ± 4.0	303.8 ± 18.0	12 ± 0.0	76 ± 5.6	309.5 ± 6.7	30 ± 0.1
500	72 ± 0.2	303.6 ± 1.0	0 ± 0.0	78 ± 0.6	313.0 ± 10.0	0 ± 0.0	59 ± 1.7	307.3 ± 8.0	13 ± 0.0	40 ± 0.9	306.9 ± 4.7	24 ± 0.0

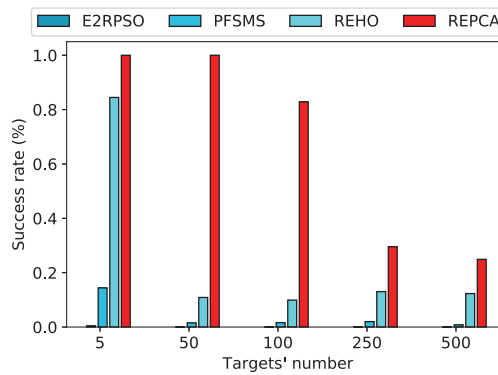


FIGURE 16. Success rate comparison with different initial number of targets in a random RC.

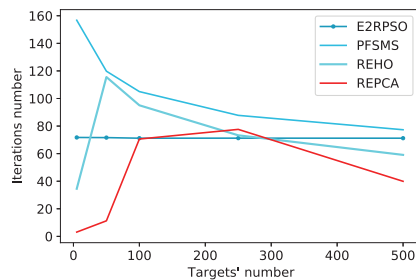


FIGURE 17. Iterations number comparison with different initial targets number in a random RC.

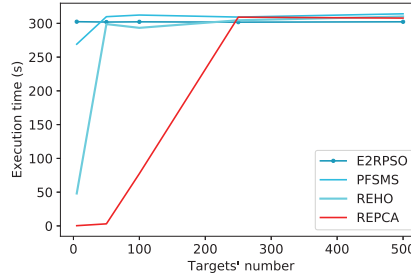


FIGURE 18. Execution time comparison with different initial number of targets in a random RC.

TABLE 7. Average iterations number, execution time, and success rate with different initial reproduction rates.

$R_0$	E2RPSO			PFSMS			REHO			REPCA		
	Iter	Time	Succ	Iter	Time	Succ	Iter	Time	Succ	Iter	Time	Succ
Growing CR												
1.5	72 ± 0.7	302.4 ± 1.3	0 ± 0.0	155 ± 2.0	308.7 ± 5.2	2 ± 0.0	143 ± 9.1	307.3 ± 4.3	13 ± 0.0	92 ± 194.8	47.8 ± 101.0	93 ± 0.2
2	73 ± 0.4	302.9 ± 1.7	0 ± 0.0	106 ± 1.6	315.4 ± 10.7	1 ± 0.0	92 ± 2.9	294.3 ± 26.8	7 ± 0.0	62 ± 113.1	48.6 ± 103.0	90 ± 0.2
2.5	72 ± 0.6	302.4 ± 1.6	0 ± 0.0	106 ± 1.3	313.9 ± 10.0	1 ± 0.0	90 ± 3.3	286.5 ± 36.3	7 ± 0.0	64 ± 99.4	58.3 ± 111.9	88 ± 0.3
3	71 ± 0.6	302.1 ± 1.1	0 ± 0.0	86 ± 1.1	310.1 ± 6.0	1 ± 0.0	73 ± 3.9	286.2 ± 47.8	7 ± 0.0	73 ± 102.0	80.2 ± 131.0	81 ± 0.3
3.5	71 ± 0.6	302.4 ± 1.2	0 ± 0.0	86 ± 1.2	310.7 ± 6.5	1 ± 0.0	75 ± 3.0	299.9 ± 29.6	7 ± 0.0	106 ± 104.4	141.2 ± 152.0	65 ± 0.4
Random CR												
1.5	71 ± 0.3	300.7 ± 1.3	0 ± 0.0	157 ± 3.1	309.8 ± 3.7	2 ± 0.0	145 ± 13.0	305.7 ± 3.4	13 ± 0.0	70 ± 156.0	37.7 ± 89.4	95 ± 0.1
2	71 ± 0.0	300.2 ± 0.1	0 ± 0.0	106 ± 1.2	313.2 ± 9.1	1 ± 0.0	91 ± 2.7	296.3 ± 28.7	7 ± 0.0	37 ± 73.5	27.9 ± 75.2	95 ± 0.2
2.5	71 ± 0.0	300.3 ± 0.1	0 ± 0.0	104 ± 1.6	309.4 ± 6.0	1 ± 0.0	90 ± 3.2	288.7 ± 33.4	7 ± 0.0	32 ± 51.9	28.2 ± 76.1	95 ± 0.2
3	72 ± 0.3	303.0 ± 0.9	0 ± 0.0	86 ± 0.9	309.2 ± 6.2	1 ± 0.0	75 ± 4.5	296.2 ± 57.5	4 ± 0.0	101 ± 117.8	121.6 ± 149.7	70 ± 0.4
3.5	72 ± 0.0	303.4 ± 0.2	0 ± 0.0	86 ± 0.8	310.5 ± 7.1	1 ± 0.0	75 ± 3.1	302.1 ± 35.9	6 ± 0.0	50 ± 72.7	59.3 ± 115.1	86 ± 0.3
Diminishing CR												
1.5	72 ± 0.0	302.6 ± 0.1	0 ± 0.0	156 ± 2.4	309.4 ± 5.1	2 ± 0.0	142 ± 15.6	305.9 ± 5.1	13 ± 0.0	36 ± 98.2	17.6 ± 53.2	98 ± 0.1
2	72 ± 0.0	302.7 ± 0.1	0 ± 0.0	106 ± 1.0	309.3 ± 8.8	1 ± 0.0	91 ± 3.5	284.8 ± 31.6	7 ± 0.0	110 ± 164.7	88.8 ± 134.4	81 ± 0.3
2.5	72 ± 0.2	302.8 ± 0.1	0 ± 0.0	105 ± 1.7	310.0 ± 6.2	1 ± 0.0	90 ± 3.5	284.9 ± 32.5	8 ± 0.0	64 ± 104.7	57.9 ± 112.3	88 ± 0.3
3	72 ± 0.0	303.4 ± 0.1	0 ± 0.0	87 ± 1.3	310.7 ± 5.8	1 ± 0.0	75 ± 2.2	305.6 ± 20.6	6 ± 0.0	44 ± 66.0	50.1 ± 107.4	89 ± 0.3
3.5	72 ± 0.0	303.4 ± 0.1	0 ± 0.0	86 ± 0.9	312.0 ± 7.3	1 ± 0.0	74 ± 3.9	294.5 ± 44.1	7 ± 0.0	56 ± 70.9	79.5 ± 130.6	82 ± 0.3

### 5.4. Influence of the reproduction rate

This subsection concerns the study of the impact of the initial reproduction rate  $R_0$  on the different resolution methods and within three kinds of Containment Rate evolution types: growing CR, random CR, and diminishing CR. The initial reproduction rate has been varied within  $\langle 1.5, 2, 2.5, 3, 3.5 \rangle$  depending on the studied variant. The other parameters have been fixed. The initial targets' number “ $\#Target_0$ ” equals 100 targets, and the population size is set to 40 individuals. Table 7 illustrates the obtained results. It appears that the difference between the three studied modes is not significant. We can deduce that the initial reproduction rate  $R_0$  is much more impacting than the CR variation strategy. So, with the rise of the reproduction rate, the reached results drop-down at different speeds for the four algorithms.

The graphic representations of the average success rate, iterations number, and execution time in a random CR are represented in Figures 19–21. It shows that E2RPSO, PFSMS, and REHO struggle to find some targets, while REPCA obtains relatively good results even with a high  $R_0$ . The execution time of the three comparison methods reaches 300 s and end-up aborting the mission in most executions. Contrary to the proposed algorithm, which achieves a success rate of over 86% in very reasonable timing.

These findings highlight REPCAs strong potential in real-world scenarios where rapid response and high efficiency are essential, such as in pandemic containment or disaster relief. While other algorithms (E2RPSO,

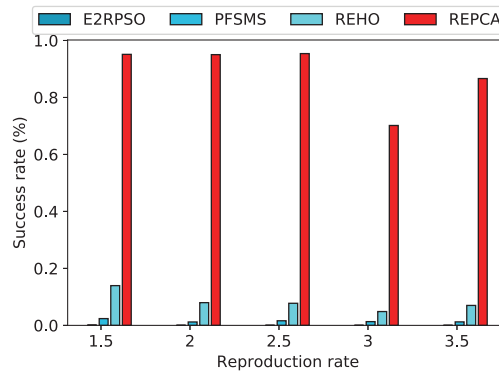


FIGURE 19. Success rate comparison with different initial reproduction rates in a random RC.

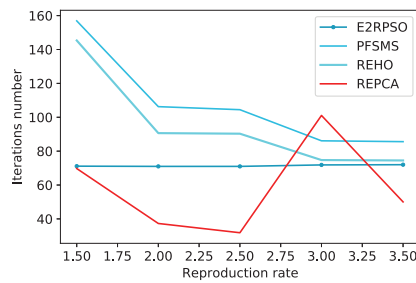


FIGURE 20. Iterations number comparison with different initial reproduction rates in random RC.

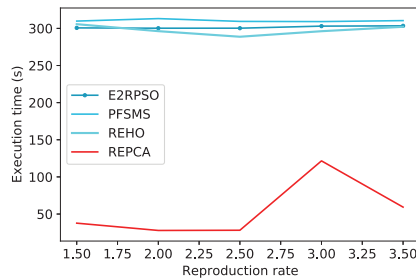


FIGURE 21. Execution time comparison with different initial reproduction rates in a random RC.

PFSMS, REHO) frequently fail to complete their missions, REPCA consistently achieves over 86% success within a reasonable time.

This performance is particularly important in the context of infectious diseases with high reproduction rates ( $R_0$ ), where the virus spreads rapidly and containment must be swift and effective. REPCA's ability to maintain high success rates even under such challenging conditions shows strong adaptability and robustness.

In practice, this means REPCA can better support robotic teams in quickly locating and addressing infection zones, reducing delays and mission failures. Its efficiency makes it highly suitable for dynamic, high-pressure environments.

## 5.5. Discussion

As shown through the experimental results, the proposed REPCA method proved its efficiency within different environmental parameters. REPCA outperforms the original REHO, even if REHO was already performing well. The hybridization with PSO and the Cultural dimension enabled it to get better results. The PFSMS approach performs well within the classical TDP, but when it comes to the Dynamic TDP that mimics the Covid-19 spread, it has some difficulties reaching good performances. Finally, despite the improvements made in E2RPSO for multi-target search, the algorithm requires a tremendous number of iterations to achieve its goals. As presented in [16], it requires around 3000 iterations to find 10 targets, which is far from our requirement of finding a dynamically growing number of targets initialized to 5, 100, or even 500, especially since we stop the search process after 300 s.

### 5.5.1. REPCA strengths

It is important to emphasize REPCAs robustness under extreme conditions, especially in scenarios modeled after high-spread diseases like COVID-19. Although the paper addresses rapid target proliferation through the DTDP framework, highlighting how REPCA maintains high success and efficiency despite increasing complexity would reinforce its value. This resilience underscores REPCAs suitability for real-world, fast-evolving crises where other methods fail.

### 5.5.2. REPCA limitations

In addition to its strengths, the REPCA algorithm has some limitations that should be acknowledged. While it performs well in simulations, its scalability in real-time systems may be limited by computational demands, especially in large-scale deployments. REPCA may also rely heavily on inter-agent communication, which can be disrupted in environments with poor connectivity. Its performance could be sensitive to initial robots and target distributions, potentially leading to suboptimal coverage. REPCA may also face challenges in dynamic scenarios with moving targets or obstacles, where its current mechanisms may not adapt quickly enough. Addressing these constraints is crucial for enhancing the algorithms robustness and broadening its applicability.

## 6. CONCLUSION AND FUTURE WORKS

This paper addresses the Dynamic Target Detection Problem (DTDP) by hybridizing two branches of Artificial Intelligence: Machine Learning and Swarm Intelligence. First, two ML techniques (GPU-kNN and DL) were used to emulate the REHO strategy as a form of cultural inheritance of robotic behavior. Among them, DL-REHOI proved to be the most faithful and effective imitation. The second phase integrated DL-REHOI with the PSO algorithm and a cultural algorithm, adding a collective learning dimension to the search strategy. Extensive experiments on 75 datasets with three different containment rates demonstrated the superiority of the proposed REPCA approach over recent state-of-the-art methods. Notably, REPCA maintained high performance and reliability even under scenarios simulating extreme conditions such as the rapid spread of COVID-19 and increasing numbers of dynamic targets. It exceeded REHO's success rate by at least 233%, particularly excelling in execution time. These findings highlight the critical role of cultural memory and PSOs inertia-based balancing of exploration and exploitation in enhancing robot adaptability. Overall, the results confirm that incorporating cultural learning and diversification mechanisms significantly strengthens the algorithms resilience in highly dynamic environments.

Among the main findings of this study, it became apparent that providing the robots with a cultural dimension enhanced their behavior against DTDP. PSO's weight inertia parameters enhance the diversification/intensification balance of the robots throughout the search process. Finally, we found that approaches with boosted diversification perform better on the DTDP.

In the future, we will try to handle the main drawback of our approach, especially, the execution time, by proposing a GPU version of the REPCA. In this parallelized version, each robot will work in an independent thread and each clan will be grouped in a block. Moreover, inspired by M. Ramezani *et al.*'s work [56], we

intend to improve the population initialization of the REPCA by providing it with an opposition-based learning method. We also plan to use the proposed method to optimize other tasks in swarm robotics.

#### ACKNOWLEDGMENTS

We would like to express our special gratitude to the Directorate General for Scientific Research and Technological Development (DGRSDT) for supporting this work under the grant number C0662300.

#### CONFLICTS OF INTEREST

The authors declare no potential conflict of interest.

#### DATA AVAILABILITY STATEMENT

Data sharing does not apply to this article. The used datasets are randomly generated during the execution. All the necessary details to generate them are given in this paper.

#### AUTHOR CONTRIBUTION STATEMENT

Naila Aziza HOUACINE: Conceptualization, Methodology, Software, Experimentation, Formal analysis, Investigation, Writing, and editing. Habiba Drias: Conceptualization, Methodology, Resources, Supervision, reviewing.

#### REFERENCES

- [1] T.K. Shackelford and V.A. Weekes-Shackelford, editors, *Cultural Learning*. Springer International Publishing, Cham (2021) 1674–1676.
- [2] J. Peedicayil, *Cultural Inheritance*. Springer International Publishing, Cham (2018) 1–4.
- [3] S. Dargan, M. Kumar, M.R. Ayyagari and G. Kumar, A survey of deep learning and its applications: a new paradigm to machine learning. *Arch. Comput. Methods Eng.* **27** (2019) 1071–1092.
- [4] J. Han, M. Kamber and P. Jian, *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco (2012).
- [5] A. Slowik, *Swarm Intelligence Algorithms: A Tutorial*. Taylor & Francis Group (2020).
- [6] A. Chakraborty and A.K. Kar, Swarm intelligence: a review of algorithms, in *Nature-Inspired Computing and Optimization*. Springer International Publishing, Cham (2017) 475–494.
- [7] G. Zhang, C. Xiao and N. Razmjoooy, Optimal parameter extraction of PEM fuel cells by meta-heuristics. *Int. J. Ambient Energy* **43** (2020) 2510–2519.
- [8] M. Mir, M. Dayyani, T. Sutikno, M.M. Zanjireh and N. Razmjoooy, Employing a Gaussian particle swarm optimization method for tuning multi input multi output-fuzzy system as an integrated controller of a micro-grid with stability analysis. *Comput. Intell.* **36** (2019) 225–258.
- [9] N. Razmjoooy, M. Ramezani and N. Ghadimi, Imperialist competitive algorithm-based optimization of neuro-fuzzy system parameters for automatic red-eye removal. *Int. J. Fuzzy Syst.* **19** (2017) 1144–1156.
- [10] N. Razmjoooy, M. Khalilpour and M. Ramezani, A new meta-heuristic optimization algorithm inspired by FIFA world cup competitions: theory and its application in PID designing for AVR system. *J. Control Autom. Electr. Syst.* **27** (2016) 419–440.
- [11] C. Khelfa and I. Khennak, A survey on recent optimization strategies in ambulance dispatching and relocation problems, in *Artificial Intelligence Doctoral Symposium*. Springer Nature, Singapore (2023) 192–203.
- [12] Y. Drias and H. Drias, Fuzzy elephant herding optimization and dbscan for emergency transportation: a case study for the 2023 türkiye earthquake, in *Intelligent and Fuzzy Systems*, edited by C. Kahraman, S. Cevik Onar, S. Cebi, B. Oztaysi, A.C. Tolga and I. Ucal Sari. Vol. 1089. Springer, Cham (2024) 403–410.
- [13] N.A. Houacine and H. Drias, When robots contribute to eradicate the COVID-19 spread in a context of containment. *Prog. Artif. Intell.* **10** (2021) 391–416.
- [14] M. Senanayake, I. Senthoran, J.C. Barca, H. Chung, J. Kamruzzaman and M. Murshed, Search and tracking algorithms for swarms of robots: a survey. *Robot. Auton. Syst.* **75** (2016) 422–434.
- [15] M. Dadgar, S. Jafari and A. Hamzeh, A PSO-based multi-robot cooperation method for target searching in unknown environments. *Neurocomputing* **177** (2016) 62–74.
- [16] J. Yang, R. Xiong, X. Xiang and Y. Shi, Exploration enhanced RPSO for collaborative multitarget searching of robotic swarms. *Complexity* **2020** (2020) 1–12.

- [17] V. Garg, A. Shukla and R. Tiwari, AERPSO – an adaptive exploration robotic pso based cooperative algorithm for multiple target searching. *Expert Systems with Applications* **209** (2022) 118245.
- [18] R. Mottaghi and R. Vaughan, An integrated particle filter and potential field method applied to cooperative multi-robot target tracking. *Auton. Robot.* **23** (2007) 19–35.
- [19] G. Li, S. Tong, F. Cong, A. Yamashita and H. Asama, Improved artificial potential field-based simultaneous forward search method for robot path planning in complex environment, in 2015 IEEE/SICE International Symposium on System Integration (SII). IEEE (2015).
- [20] J. Li and Y. Tan, Triangle formation based multiple targets search using a swarm of robots, in *Lecture Notes in Computer Science*. Springer International Publishing, Cham (2016) 544–552.
- [21] J. Li and Y. Tan, A probabilistic finite state machine based strategy for multi-target search using swarm robotics. *Appl. Soft Comput.* **77** (2019) 467–483.
- [22] J. Li and Y. Tan, A two-stage imitation learning framework for the multi-target search problem in swarm robotics. *Neurocomputing* **334** (2019) 249–264.
- [23] K.N. Krishnanand and D. Ghose, A glowworm swarm optimization based multi-robot system for signal source localization, in *Studies in Computational Intelligence*. Springer, Berlin (2009) 49–68.
- [24] K. McGill and S. Taylor, Comparing swarm algorithms for multi-source localization, in 2009 IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR 2009). IEEE (2009).
- [25] M.S. Couceiro, P.A. Vargas, R.P. Rocha and N.M. Ferreira, Benchmark of swarm robotics distributed techniques in a search task. *Robot. Auton. Syst.* **62** (2014) 200–213.
- [26] Z. Zheng, J. Li, J. Li and Y. Tan, Improved group explosion strategy for searching multiple targets using swarm robotics, in 2014 IEEE International Conference on Systems, Man and Cybernetics (SMC). IEEE (2014).
- [27] H. Tang, W. Sun, H. Yu, A. Lin, M. Xue and Y. Song, A novel hybrid algorithm based on PSO and FOA for target searching in unknown environments. *Appl. Intell.* **49** (2019) 2603–2622.
- [28] A. Iriundo, E. Lazkano, A. Ansuategi, A. Rivera, I. Lluvia and C. Tubo, Learning positioning policies for mobile manipulation operations with deep reinforcement learning. *Int. J. Mach. Learn. Cybern.* **14** (2023) 3003–3023.
- [29] Y. Zhao, B. Chen, X. Wang, Z. Zhu, Y. Wang, G. Cheng, R. Wang, R. Wang, M. He and Y. Liu, A deep reinforcement learning based searching method for source localization. *Inf. Sci.* **588** (2022) 67–81.
- [30] Q. Luo, T.H. Luan, W. Shi and P. Fan, Deep reinforcement learning based computation offloading and trajectory planning for multi-UAV cooperative target search. *IEEE J. Sel. Areas Commun.* **41** (2023) 504–520.
- [31] M. Zhao, H. Lu, S. Yang and F. Guo, The experience-memory q-learning algorithm for robot path planning in unknown environment. *IEEE Access* **8** (2020) 47824–47844.
- [32] Z. Bai, H. Pang, Z. He, B. Zhao and T. Wang, Path planning of autonomous mobile robot in comprehensive unknown environment using deep reinforcement learning. *IEEE Internet Things J.* **11** (2024) 22153–22166.
- [33] K. Arulkumaran, M.P. Deisenroth, M. Brundage and A.A. Bharath, Deep reinforcement learning: a brief survey. *IEEE Signal Process. Mag.* **34** (2017) 26–38.
- [34] G. Dulac-Arnold, N. Levine, D.J. Mankowitz, J. Li, C. Paduraru, S. Gowal and T. Hester, Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Mach. Learn.* **110** (2021) 2419–2468.
- [35] G.G. Wang, L.D.S. Coelho, X.Z. Gao and S. Deb, A new metaheuristic optimisation algorithm motivated by elephant herding behaviour. *Int. J. Bio-Inspired Comput.* **8** (2016) 394.
- [36] N.A. Houacine and H. Drias, DISS: a discrete input-space sampling path planning and obstacle avoidance strategy for swarm robotics, in *Artificial Intelligence Doctoral Symposium*. Springer Nature, Singapore (2023) 148–161.
- [37] D. Soydaner, A comparison of optimization algorithms for deep learning. *Int. J. Pattern Recognit. Artif. Intell.* **34** (2020) 2052013.
- [38] A. Nguyen, K. Pham, D. Ngo, T. Ngo and L. Pham, An analysis of state-of-the-art activation functions for supervised deep neural network, in 2021 International Conference on System Science and Engineering (ICSSE) (2021) 215–220.
- [39] Y.A. LeCun, L. Bottou, G.B. Orr and K.-R. Müller, Efficient BackProp, in *Lecture Notes in Computer Science*. Springer, Berlin (2012) 9–48.
- [40] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, edited by Y.W. Teh and M. Titterton. Vol. 9 of *Proceedings of Machine Learning Research*. Chia Laguna Resort, Italy (2010) 249–256.
- [41] K. He, X. Zhang, S. Ren and J. Sun, Delving deep into rectifiers: surpassing human-level performance on ImageNet classification, in 2015 IEEE International Conference on Computer Vision (ICCV). IEEE (2015).
- [42] L.N. Smith, Cyclical learning rates for training neural networks, in 2017 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE (2017).

- [43] Q. Kuang and L. Zhao, A practical GPU based kNN algorithm, in Proceedings of the Second Symposium on International Computer Science and Computational Technology (ISCCT'09). Academy Publisher (2009) 151–155.
- [44] D.W. Aha, Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *Int. J. Man-Mach. Stud.* **36** (1992) 267–287.
- [45] A.C. Cameron and F.A. Windmeijer, An r-squared measure of goodness of fit for some common nonlinear regression models. *J. Econom.* **77** (1997) 329–342.
- [46] S. Cheng, Y. Shi, Q. Qin, Q. Zhang and R. Bai, Population diversity maintenance in brain storm optimization algorithm. *J. Artif. Intell. Soft Comput. Res.* **4** (2014) 83–97.
- [47] K. Hussain, M.N.M. Salleh, S. Cheng and Y. Shi, On the exploration and exploitation in popular swarm-based metaheuristic algorithms. *Neural Comput. Appl.* **31** (2018) 7665–7683.
- [48] J. Kennedy and R. Eberhart, Particle swarm optimization, in Proceedings of ICNN 95 – International Conference on Neural Networks. IEEE (1994).
- [49] J.C. Bansal, P.K. Singh, M. Saraswat, A. Verma, S.S. Jadon and A. Abraham, Inertia weight strategies in particle swarm optimization, in 2011 Third World Congress on Nature and Biologically Inspired Computing. IEEE (2011).
- [50] A. Rathore and H. Sharma, Review on inertia weight strategies for particle swarm optimization, in *Advances in Intelligent Systems and Computing*. Springer, Singapore (2017) 76–86.
- [51] J. Xin, G. Chen and Y. Hai, A particle swarm optimizer with multi-stage linearly-decreasing inertia weight, in 2009 International Joint Conference on Computational Sciences and Optimization. IEEE (2009).
- [52] R.G. Reynolds, An introduction to cultural algorithms, in Evolutionary Programming – Proceedings of the Third Annual Conference, edited by A.V. Sebald and L.J. Fogel. World Scientific Press, Singapore (1994) 131–139.
- [53] N.A. Houacine and H. Drias, Self-parameterized swarm intelligence algorithms for targets' detection in complex and unknown environments, in Hybrid Intelligent Systems. Springer International Publishing, Cham (2021) 690–699.
- [54] A. Bruce, L. Wannian, D. Xiaoping, E. Tim, F. Dale, I. Chikwe, L. Clifford, L. Jong-Koo, L. Gabriel, L. Jiangtao, L. Haiying, P. Natalia, S. Aleksandr, T. Hitoshi, V.K. Maria, W. Bin, W. Guangfa, W. Fan, W. Zhongze, W. Zunyou, X. Jun, Y. Kwok-Yung, Z. Weigong, Z. Yong and Z. Lei, Report of the who-china joint mission on coronavirus disease 2019 (covid-19). Technical Report. World Health Organization, China (2020).
- [55] Comment lpidmiologie tente de cerner lpidmie due au nouveau coronavirus. *Le Monde* (2020).
- [56] M. Ramezani, D. Bahmanyar and N. Razmjoo, A new improved model of marine predator algorithm for optimization problems. *Arab. J. Sci. Eng* **46** (2021) 8803–8826.



**Please help to maintain this journal in open access!**

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting [subscribers@edpsciences.org](mailto:subscribers@edpsciences.org).

More information, including a list of supporters and financial transparency reports, is available at <https://edpsciences.org/en/subscribe-to-open-s2o>.