Particle Swarm Algorithm Setting using Deep Reinforcement Learning in the Artificial Neural Network Optimization Learning Process

Abdelkader Fassi Fihri, Tarik Hajji, *Member, IAENG,* Ibtissam El Hassani, and Tawfik Masrour, *Member, IAENG*

***Abstract*—Recent meta-heuristics for optimization issues include particle swarm optimization (PSO) techniques. They take their cues from the coordinated movements of swarms of fish and birds, which exhibit collective behaviors. A sophisticated learning phase, like back-propagation, is needed for artificial neural networks (ANNs), which are thought of as a source of artificial intelligence (AI). This phase allows for the calculation of the error gradient for each neuron, from the final layer to the first.**

However, some qualities of the objective function are necessary (cost). This inspired us to experiment with meta-heuristics to streamline the training of ANNs to manage complex nonlinear systems.

The objective of this research is to apply deep reinforcement learning (DRL) to automatically calculate the PSO algorithm's parameters while also optimizing the supervised learning process of ANN. After a number of case studies, our methodology consistently leads to the coefficients of the ideal ANN.

***Index Terms*—IA, ANN, CNN, PSO, machine learning, deep learning, deep reinforcement learning, meta-heuristics, and optimization.**

I. Introduction

OUR team has worked on several issues related to the use of AI in various industrial contexts! The integration of artificial intelligence in the industry can bring tremendous benefits in terms of efficiency, productivity, and decision-making [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. A branch of mathematics known as optimization finds the best element in a set by comparing it to a set of predetermined criteria. As a result, optimization is ubiquitous and has long been undergoing continual development [11], [12], [13].

Manuscript received February 16, 2023; revised July 15, 2024. This work was supported in part by Moulay Ismail University, Meknes, Morroco.

Abdelkader Fassi Fihri is Professor at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Mathematical Modeling, Analysis and Simulation team (M2AS), Department of Mathematics and Computer Science, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: a.fassifihri@ensam.umi.ac.ma

Tarik Hajji is Professor at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Artificial Intelligence for Engineering Sciences Team (IASI), Department of Mathematics and Computer Science, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: t.hajji@umi.ac.ma

Ibtissam EL Hassani is Professor at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Artificial Intelligence for Engineering Sciences Team (IASI), Department of Mathematics and Computer Science, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: i.elhassani@ensam.umi.ac.ma

Tawfik Masrour is Professor at Laboratory of Mathematical Modeling, Simulation and Smart Systems (L2M3S), Artificial Intelligence for Engineering Sciences Team (IASI), Department of Mathematics and Computer Science, 15290 ENSAM, Moulay Ismail University, 50500 Meknes, Morocco, e-mail: t.masrour@ensam.umi.ac.ma

In the research space, optimization techniques look for a solution or set of solutions that fulfill the set of constraints and reduce or maximize the objective function [14], [15]. Meta heuristics, one of these techniques, are generic optimization algorithms that aim to enable the solution of a variety of different problems without necessitating substantial changes to the algorithm [16]. They collectively make up a family of algorithms designed to tackle challenging optimization issues for which there is no known more efficient conventional approach [17]. This group of techniques includes PSO, which is derived from stochastic descent. The gregarious interactions of migrating animals, such as those shown in Fig. 1, which must travel long distances and thus maximize their motions in terms of energy expended, such as the "V formation", are a major source of inspiration [18].

The area of modern IA is strongly related to the ANN technique. This takes its cues from the sophisticated information processing that occurs in the human brain [7]. This behavior is supported by a variety of mental mechanisms that are based on neurophysiological (branch of physiology dealing with the functions of the nervous system) processes. This method is capable of accounting for the "non-linearity" and uncertainties present in actual systems. It is built on these systems' capacities for learning and optimization [19], [3].

The most popular optimization technique for training ANNs is "back-propagation." It enables the determination of the error gradient for each neuron, from the deepest layer to the surface layer [5]. However, it requires specific qualities from the objective function. Due to this, we decided to test the PSO "meta-heuristic" method for enhancing ANN learning for complicated nonlinear system control.

PSO, like other algorithms, requires the definition of a set of parameters, which we have established using DRL [20], which is a branch of machine learning (ML) where the learner, called an agent, interacts autonomously with the environment to acquire the necessary skills. In each phase, the agent is meant to select the action that produces the highest reward signal. The rest of this paper is organized as follows: The state of the art section presents the principles of PSO as well as related works in a general sense; The methodology section explains our optimization approach. The case study section presents the application of the optimization approach on the training of an ANN in a supervised classification problem; The interpretation and a comparison with other optimization techniques are presented in the discussion section, and the part that follows concludes.

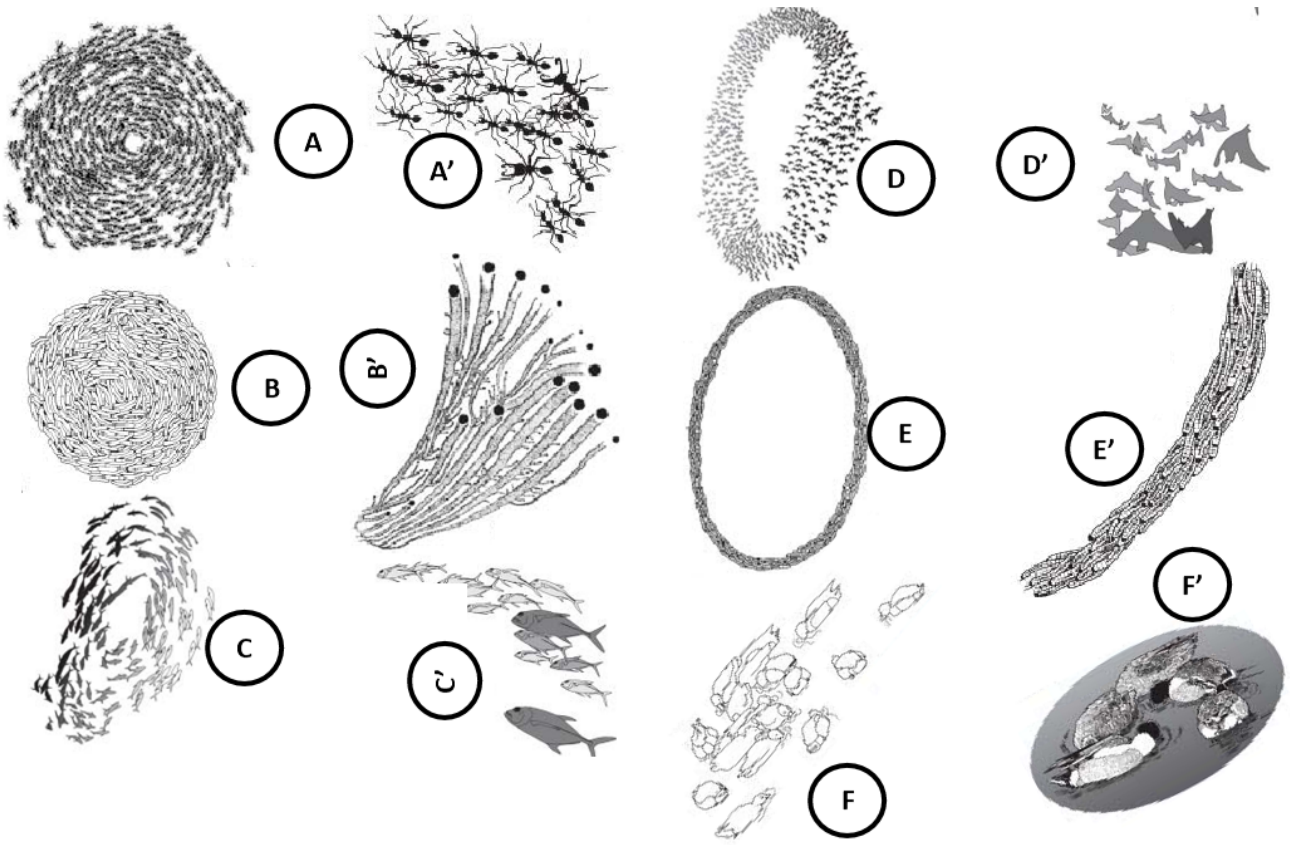

Fig. 1. PSO (Particle Swarm Optimization): PSO is a heuristic optimization algorithm inspired by the social behavior of birds and insects. It is used to solve numerical optimization problems and search for optimal solutions in a search space. The PSO algorithm consists of a population of particles (or individuals) that move in the search space based on their own experiences and the collective experience of the group.

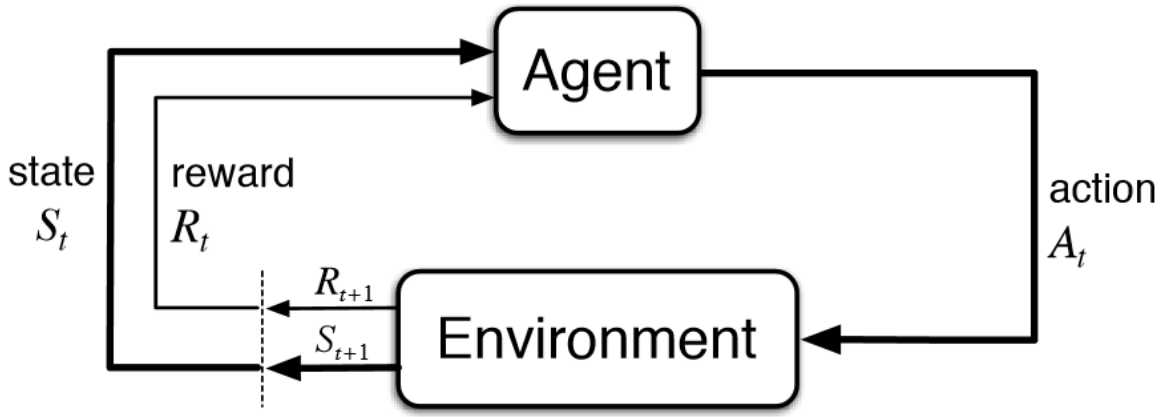

Fig. 2. The process of reinforcement learning.

II. Background

PSO is employed as an optimization technique in a number of works to address complicated issues. In [1], Wang and colleagues present a method for assessing various convolutional neural network (CNN) topology that makes use of PSO. Their aim is to automatically find the best CNN architecture for image classification issues. The neural architecture search (NAS) algorithm is replaced by the suggested method, which has the drawback of being slow. This intriguing research area has the potential to replace expert-designed networks with learned task-specific architectures [2].

In [3], Beatriz and colleagues present a methodology that automatically designs an ANN using PSO, such as Basic PSO, Second Generation of PSO and a new model of PSO called "NMPSO." The aim of these algorithms is to evolve, at the same time, the three principal components of an ANN: the set of synaptic weights; the connections, or architecture; and the transfer functions for each neuron.

In [4], the authors explain how PSO is applied to search for global solutions for reinforcement learning problems. This particle swarm optimization policy (PSO-P) is efficient for high-dimensional state spaces and does not require a priory

assumptions about the proper representations of the policy.

Bashir and colleagues demonstrate in [5] the significant potential of the PSO technique to optimize parameter settings and hence conserve valuable computational resources during the tuning of deep learning models. "Ada-Swarm," a revolutionary gradient-free optimizer with performance comparable to or superior to that of the "Adam" optimizer used in neural networks, is introduced in [6].

A. Particle Swarm Algorithm

In many real-world applications, there is always a need to find optimal configurations from a discrete set of objects; this is known as a combinatorial optimization problem [21]. Approximations and meta heuristic algorithms have been utilized to handle these optimization problems as a trade-off between solution quality and computation time [22]. A new generation of meta-heuristic algorithms, called Particle Swarm Optimization, has been developed and has proven its efficiency [23]. Like other meta-heuristic algorithms that are inspired by nature [24], the PSO algorithm is considered an adaptive search technique based on collaboration between individuals.

PSO is a meta-heuristic optimization technique inspired by biology and invented by Eberhart and Kennedy. PSO has its origins in the behavior of groups of bees, flocks of birds, and schools of fish when searching for food [25]. Despite having limited individual capabilities, the agents that make up the group (or the particles that make up the swarm) create a form of collective intelligence [26]. Throughout the investigation, members of the same swarm interact with one another to develop a solution to the issue at hand based on their combined knowledge. As a result, thetion $f(x_1, x_2, ..., x_d)$, the objective to be optimized, $xj_{min} \leq x_j \leq x_{max}$ are continuous or discrete variables. A PSO algorithm consists of finding the combination $(x_{1opt}, x_{2opt}, ..., x_{dopt})$ that optimizes the function f (minimizes in our case). The PSO makes use of both their own experiences and the best experiences of their neighbors as they search through a space for a solution that optimizes a certain cost (or fitness) function.

Consider a function that does not require any regularity [27] in the function to be optimized (neither continuity, convexity, nor gradient calculation). It has the particularity of being simple to implement. A particle, or candidate solution, is constituted by a vector of variables of the function to be optimized. It is characterized by a velocity ν_i and a position p_i. Moving from iteration k to iteration $(k+1)$, the algorithm updates the velocity and position of each particle according to two equations (1) called equations of motion:

$$\begin{cases} \nu_{k+1} = w_k.\nu_k + b_1.r_1.(p_{best} - p_k) + b_2.r_2.(p_{gbest} - p_k) \\ p_{k+1} = p_k + \nu_{k+1} \end{cases} \tag{1}$$

where p_k is the position of each particle at iteration k, ν_k is the velocity of each particle at iteration k, w is the inertial weight of the particle, b_1 and b_2 are the constant acceleration coefficients, r_1 and r_2 are two random numbers in $[0, 1]$, p_{best} is the best position of particle until iteration k and p_{gbest} is the best position reached by the particle's neighbors.

B. The Pyswarms Library

PSO can be used to handle a wide range of issues, from straightforward optimization to robotics and workshop planning, given enough variations. Because of this, the researchers decided to develop Pyswarms [28], a research toolbox that can be used by scientists. For tackling continuous and combinatorial PSO issues, PySwarms offers a set of practical classes. It uses a black-box methodology to solve optimization problems and enables quick development of non-conventional swarm models. It also includes research tools and reference functions for assessing and enhancing swarm performance. The packages' fundamental design tenet is to strike a compromise between experimental ease and simplicity of use by offering a wide range of classes to address optimization problems and by building a consistent API to support non-standard PSO implementations. PySwarms was created using the following guiding principles:

- maintain a particular set of comprehensible conventions;
- develop a single API that all Swarm implementations can use;
- make a collection of rudimentary classes accessible for use with common PSO implementations;
- python PSO algorithms that are commonly used ;
- built-in, special-purpose testing functions;
- environmental mapping for swarm and cost animations.

PySwarms offers the following hyper-parameter search tools:

- Continuous search space;
  - $pyswarms.single.global\_best$: the classical global-best PSO algorithm with a star-topology, where each particle is compared to the best performing particle in the swarm.
  - $pyswarms.single.local\_best$: the classical local-best PSO algorithm with a ring-topology. Each particle is compared only to its nearest neighbors as calculated by a distance metric.
  - $pyswarms.single.general\_optimizer$: modifiable PSO algorithm with a custom topology. Each topology of the "pyswarms.backend" module can be considered as an argument.
- Discrete search space: Single-objective optimization with discrete search space is useful for scheduling jobs, dispatching salespeople on the road, and other sequence-based issues.
  - $pyswarms.discrete.binary$: A classic binary optimization algorithm for PSO without mutation. uses a ring topology to choose its neighbors (which can also be defined on a global scale).
- Research methods: These search methods can be used to compare the relative performance of hyper-parameter values to reduce a specific objective function.
  - $pyswarms.utils.search.grid\_search$: a thorough search for the selected goal function's best performance with regard to the Cartesian products of the given hyper-parameter values came up empty.
  - $pyswarms.utils.search.random\_search$: the classical local-best PSO algorithm with a ring-topology. Each particle is compared only to its nearest neighbors as calculated by a distance metric.

  - $pyswarms.utils.plotters$: for a defined number of selection iterations, look for the selected objective function's best performance with regard to various values of randomly chosen hyper-parameter combinations.
- Environment: various settings to evaluate swarm performance and create visualizations:
  - $py.utils.environments.plot\_environment$: a setting where particles can be animated and expenses can be tracked in a 2D or 3D space.

C. Electricity Transmission Problem

The proposed method was evaluated to increase the performance and dependability of mechanical structures, specifically to enhance load resistance while minimizing cost and maximize the lifetime of a transmission line tower ("minimizing the use of materials"). We are discussing the "Energy Tower" dilemma, in which the generation, transmission, and distribution of electricity result in losses for a variety of causes. It is advantageous to do a structural study of a transmission line tower that is similar to a planar lattice for this (Figure. 5). The two upper ends of the tower are subjected to two identical loads F of $1.8KN$ each, imposed at an angle of$= 15$. The steel bars that make up the construction have a Poisson's ratio of 0.27 and an elastic modulus of $210GPa$. Assuming that the weight of each bar in the lattice is insignificant in comparison to the applied forces, the cross-section of each bar is $A = 27.90cm2$.

The goal of this analysis is to identify the forces, stresses, and maximum displacement produced by the applied loads as well as to assess whether particular lattice elements are susceptible to buckling. To increase the dependability of the tower, design optimization will also be used to identify the ideal straight portion of each bar that minimizes the maximum movement in the bar. The variables for optimization are $X = \{H, W\}$, where H and W are the straight section of the bar's height and breadth, respectively, and these are the main constituents of the objective function, or "the function to be optimized."

D. Travelling Salesman Problem

An NP-Complete optimization problem is known as The Travelling Salesman Problem (TSP) (it is impossible to find a solution quickly, but it is fast to check the quality of a solution). The issue is to "create a path of minimum total length that passes through each city exactly once and returns to the starting point, given a set of cities separated by certain distances." We must employ techniques that enable us to derive an approximation of the solution to this problem because, despite its seeming simplicity, it cannot be solved in an acceptable amount of time for big cases. The number of potential pathways, for instance, is a $100 - digit$ figure for 69 cities. The TSP has a wide range of applications and frequently manifests as a smaller issue within a larger issue. For instance, in genetics, the concept of distance between two cities is equivalent to the degree of similarity between two DNA pieces. The issue of the traveling salesman is shown in the Figure.4.

E. Artificial Neural Networks Learning Problem

The topic of AI and ANN methodology are closely related. It is modeled after how the human brain processes information, which is considered intelligent behavior when it is backed by a set of mental procedures based on neurophysiological processes. This method is capable of accounting for the non-linearity and uncertainties present in actual systems. It is built on these systems' capacities for learning and optimization.

The most popular optimization technique for training neural networks is back propagation. It enables the computation of the error gradient for each neuron, from the topmost layer to the bottom most. However, it necessitates an understanding of the control and certain qualities in the objective function (cost). This encourages us to test the PSO meta-heuristic for neural network training optimization for complicated nonlinear systems control.

The goal is to use the PSO meta-heuristic to optimize the emulator and controller ANNs during training. This mainly entails tailoring the method to an ANN's specifications in terms of its structure, training variables, and their effects on output variation, which serves as the system's control, and comparing the method's effectiveness (performance) to that of other methods, particularly the descending gradient method.

F. Reinforcement Learning (RL)

In the branch of machine learning known as reinforcement learning, an agent interacts independently with its environment to learn how to carry out a task. In each phase, the agent is meant to select the action that produces the highest reward signal (Fig. 2). A reinforcement learning system's components are:

- $Agent$: An autonomous entity can use sensors to perceive its surroundings and defectors to act on them in order to accomplish its objectives.
- $Environment$: It is the living environment of an agent and can be discrete or continuous, episodic or sequential, deterministic or stochastic, static or dynamic, comprised of one agent or numerous agents, and can be fully or partially observable.
- $Policy$: Due to the fact that it describes the agent's behavior at a specific time, it is the fundamental component of the reinforcement learning system. A mapping of environmental situations to a set of actions, on the other hand, is what policy is.
- $Reward$: Each time the agent takes a decision, the environment notifies them with a number signal. The reward signal seeks to positively reinforce good behavior while punishing undesirable behavior. As a result, the agent gains the ability to favor positive acts in each state. In other words, the reward signal seeks to improve the policy; if the policy selects an action with a low payoff, it will be adjusted to select other actions going forward.
- $Valuefunction$: It is an estimate of the overall reward for an agent starting from state S, which is frequently given as $V(s)$. Values describe a state's long-term desirability, taking into account the expected payoff, if the reward is an urgent aim in environmental conditions. In other words, some states with low rewards have high

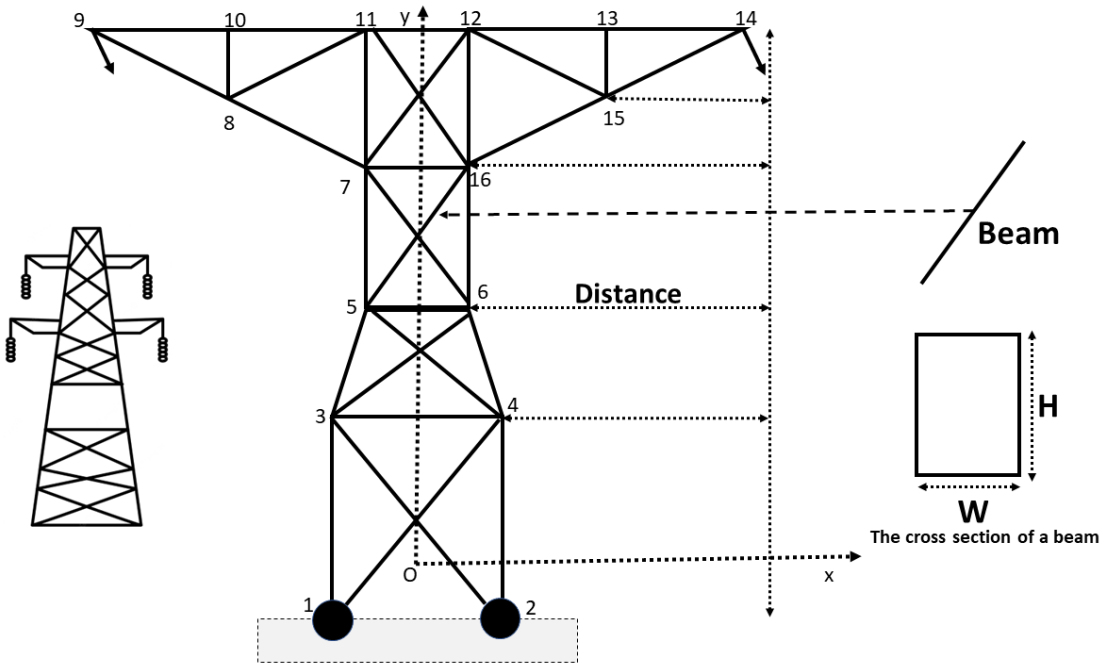

Fig. 3. Electricity pylon represented geometrically.

values, which indicates that states with high rewards follow them. Value judgments are therefore a type of forecast of rewards, and every action decision is made based on one by default.

III. METHODOLOGY

In our method, the PSO algorithm is set up to optimize the supervised learning of ANNs utilizing the DRL.

A. PSO Configuration

1) Number of particles: The allocation of particles to solve the problem is primarily influenced by two factors:

- The search area's dimensions;
- The relationship between the machine's processing power and the longest possible search time;

To choose this parameter, there is no set formula. To gain the expertise required to comprehend this parameter, it is important to conduct numerous tests. We created a DRL framework consisting of an environment and an agent to carry out this operation in order to learn how to choose this parameter based on experience:

- The ANN is the agent's surroundings.
- The ANN's learning rate serves as the foundation of the reward system.
- The parameters of the ANN represent the state space of the environment.
- The agent's options include either decreasing or increasing the parameter in relation to the incentive received.

The PSO determines the ANN's learning rate, and as a result, the reward system assesses the quality of the current parameter before changing it in an effort to make it better.

2) Neighborhood topology: The neighborhood topology defines with whom each of the particles will be able to communicate. There are many combinations of which the following are the most used:

- In star topology, every particle is interconnected with every other particle, making the neighborhood optimum the global optimum.
- In ring topology, each particle is connected to an average of n (3 particles) other particles. It is the most prevalent topology;
- In spoke topology, all particles communicate with just one central particle.

The most typical PSO topology are shown in the Figure. 6

3) Confidence coefficient: It takes into account the particle's propensities for panurgism or self-preservation. The following definition applies to the random variables $r1$ and $r2$:

$$\begin{cases} \rho_1 = b_1.r_1 \\ \rho_2 = b_2.r_2 \end{cases}$$

Where b_1 and b_2 are empirically obtained positive constants that obey the equation $b1 + b \leq 24$ and r_1 and r_2 have a uniform distribution on $[0, 1]$.

4) Maximum speed and constriction coefficient: It may be required to establish a maximum speed (denoted V_{max}) to prevent the particles from traveling too quickly through the search space and perhaps missing the optimum. This will help the algorithm's convergence. However, if we employ a constriction coefficient k that narrows the search area, this can be avoided. The velocity equation then becomes:

$$k = 1 - \rho + \sqrt{|\rho_2 - 4\rho|^2}$$

With $\rho = \rho_1 + \rho_2 > 4$

$$\nu_{k+1} = k.[w_k.\nu_k + b_1.r_1.(p_{bestk} - p_k) + b_2.r_2.(p_{pgbestk} - p_k)]$$

According to research by SHI and EBERHART, applying a constriction coefficient typically improves the convergence rate without requiring the setting of a maximum speed.

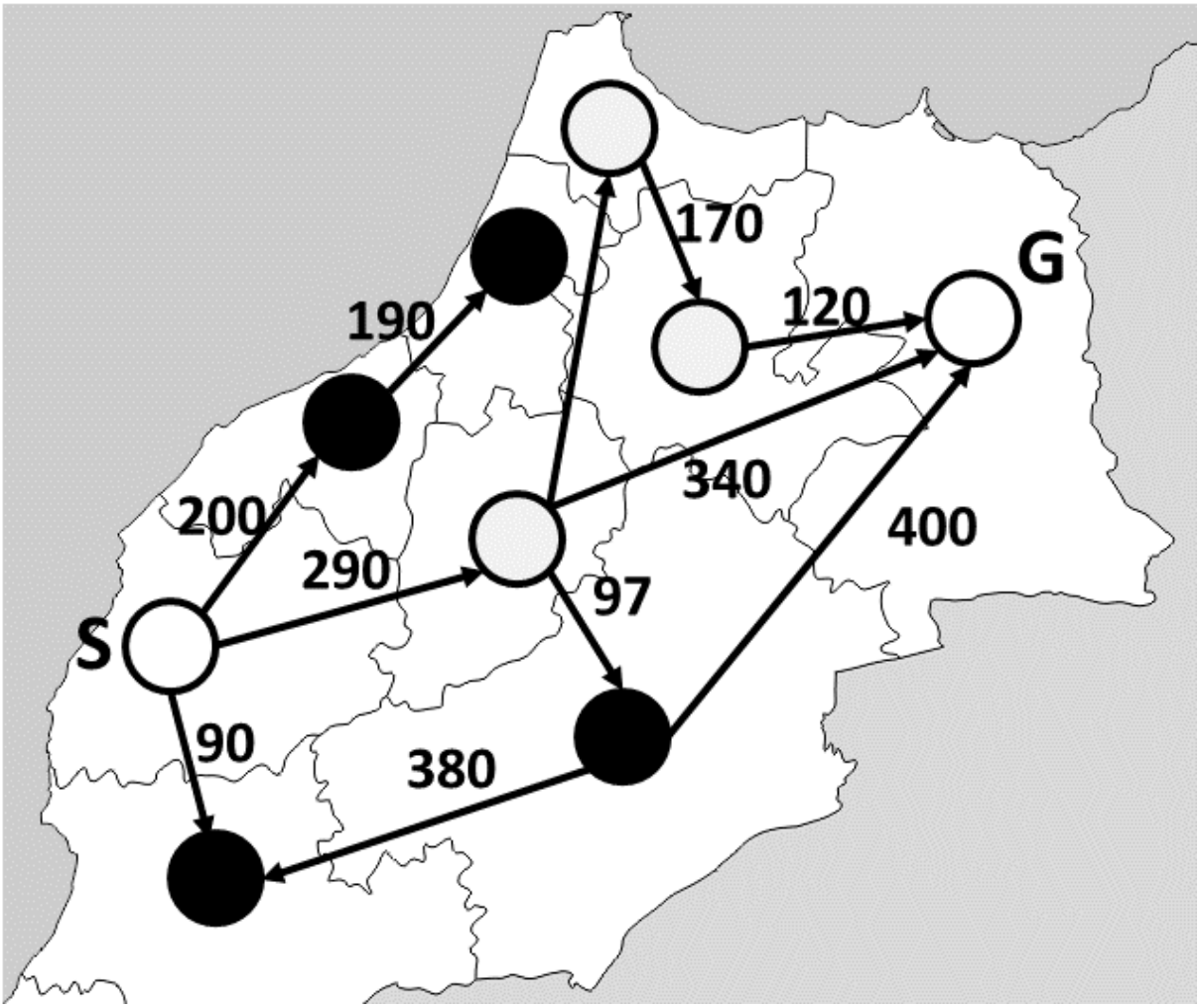

Fig. 4. Problem of the Commercial Traveller.

B. Optimization Algorithm's Steps

We consider the following notations:

- i : indices of the particles;
- j : indices of the variables to be searched;
- k: indices of the iterations;
- $fitniss\_i$: value of the fitness function of particle i;
- $persona\_best\_fitniss\_i$: best past tense realization of particle i;
- $global\_best\_futness\_i$: best fitness of the neighbors of particle i.
- Step 1:
  Define the algorithm's parameters, including the total number of particles, b_1, b_2, and iterations.
- Step 2:
  - Initialize the index of the iterations.
  - Randomly initialize the positions of the particles inside the search space.
  - Compute fitness for each particle.
  - Initialize pbest and p for each particle.
  - Find p_gbest.
- Step 3:
  - Increment the index of iterations.
  - Update the velocities and positions of the particles according to the equations (1).
  - Test the constraints: if they are violated, perform the confinement (return the particles inside the search domain).
  - Compute fitness for each particle.
  - Compare $persona\_best\_fitniss\_i$ to $fitniss\_i$: if $fitniss\_i$ is better, assign p_best to p.
  - Find the minimum on i of the $fitniss\_i$, note I this index and assign to p_gbest the value of pI.
- Step 4:
  - Test the stopping requirement; if it is met, complete the treatment; if not, go back to step 3.
  - A minimal value of the optimal cost to be reached or a predetermined number of iterations might serve as the halting criterion set in step 4 of the PSO algorithm.

Figure. 7 provides a schematic illustration of a particle's trajectory within the search space. The preceding steps are encapsulated in the following algorithm after defining the various PSO parameters:

C. PSO Application

Take into account, for illustration, the objective function used for minimization:

$$F(x) = 10(x_1 - 1)^2 + 20(x_2 - 2)^2 + 30(x_3 - 3)^2$$

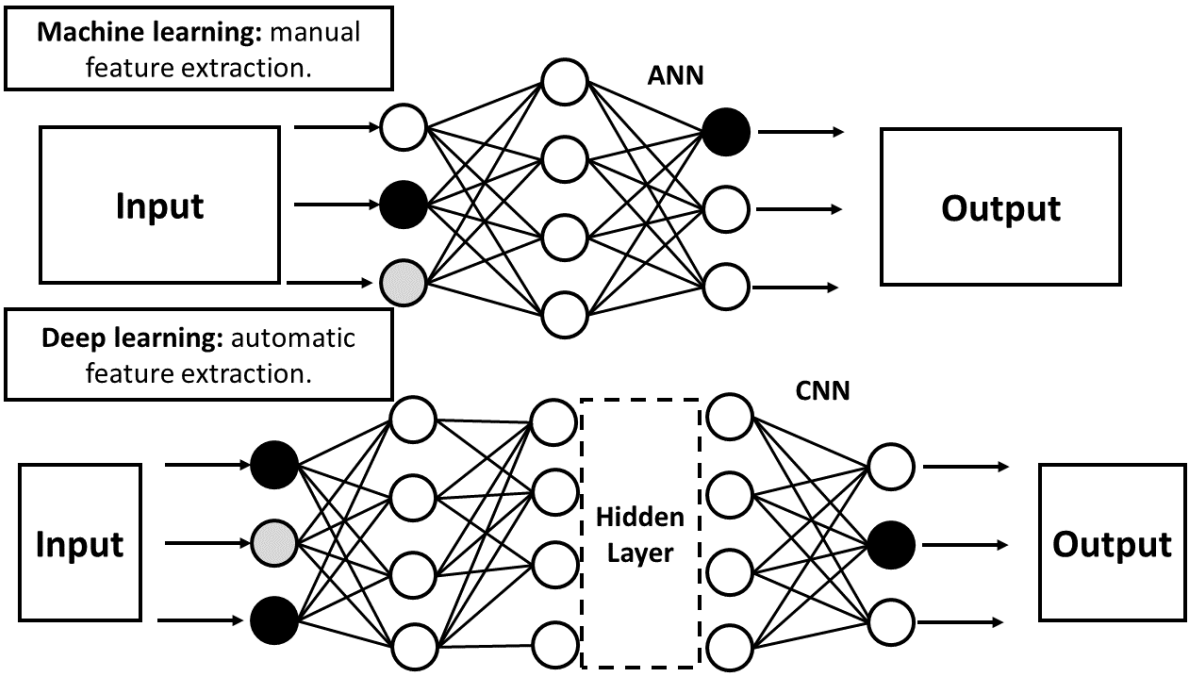

Fig. 5. Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN).

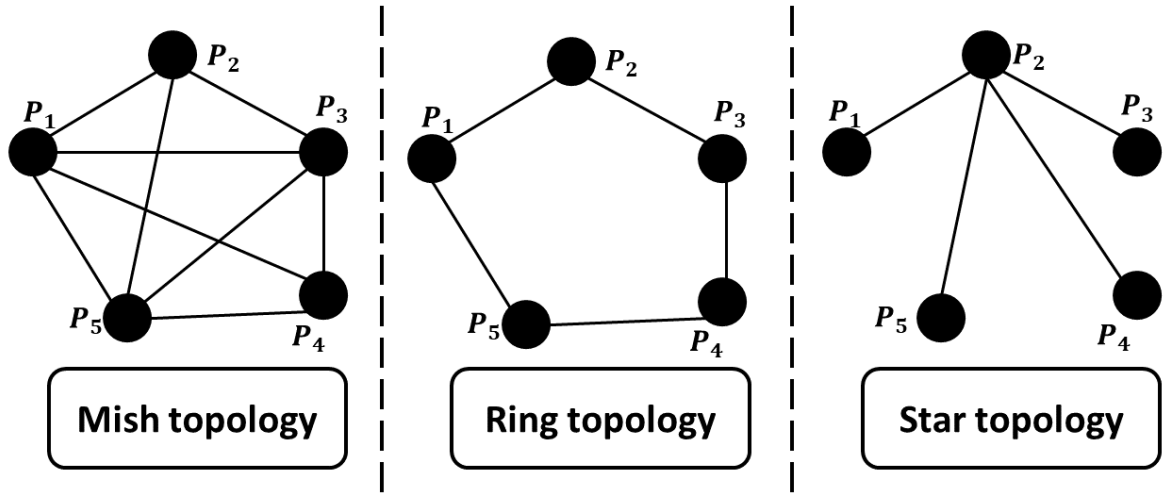

Fig. 6. Star Topology, Ring Topology and Ray Topology.

Algorithm 1 PSO Optimization Algorithm

repeat
  for i from $1toN$ **do**
    if $F(\overrightarrow{x_i}) > p_{best_i}$ **then**
      $p_{best_i} \leftarrow F(\overrightarrow{x_i})$
      $\overrightarrow{x_{pbest_i}} \leftarrow \overrightarrow{x_i}$
    end if
    if $F(\overrightarrow{x_i}(t) > g_{best_i}$ **then**
      $g_{best_i} \leftarrow F(\overrightarrow{x_i})$
      $\overrightarrow{x_{gbest}} \leftarrow \overrightarrow{x_i}$
    end if
  end for
  for $ifrom1toN$ **do**
    $\overrightarrow{\vartheta_i} \leftarrow \overrightarrow{\vartheta_i} + \rho_1(\overrightarrow{x}_{pbets_i} - \overrightarrow{x}_i) + \rho_2(\overrightarrow{x}_{gbets_i} - \overrightarrow{x}_i)$
    $\overrightarrow{x}_i \leftarrow \overrightarrow{x}_i + \overrightarrow{\vartheta_i}$
  end for
until stopping conditions

1) Initialization of parameters and population:

- Parameter initialization:
  - Number of variables: $m = 3(x_1, x_2, x_3)$;
  - Population size: $n = 5$ (we used our proposed approach that use DRL model after a learning process to determine this parameter);
  - Inertial weight for each particle: $w = 0.9$;
  - Acceleration factors for each particle: $c_1 = 2$ and $c_2 = 2$;
  - Max iteration size: $k_{max} = 50$;
- Initialization of the population:
  - Let's initialize the position (x_i) randomly for each

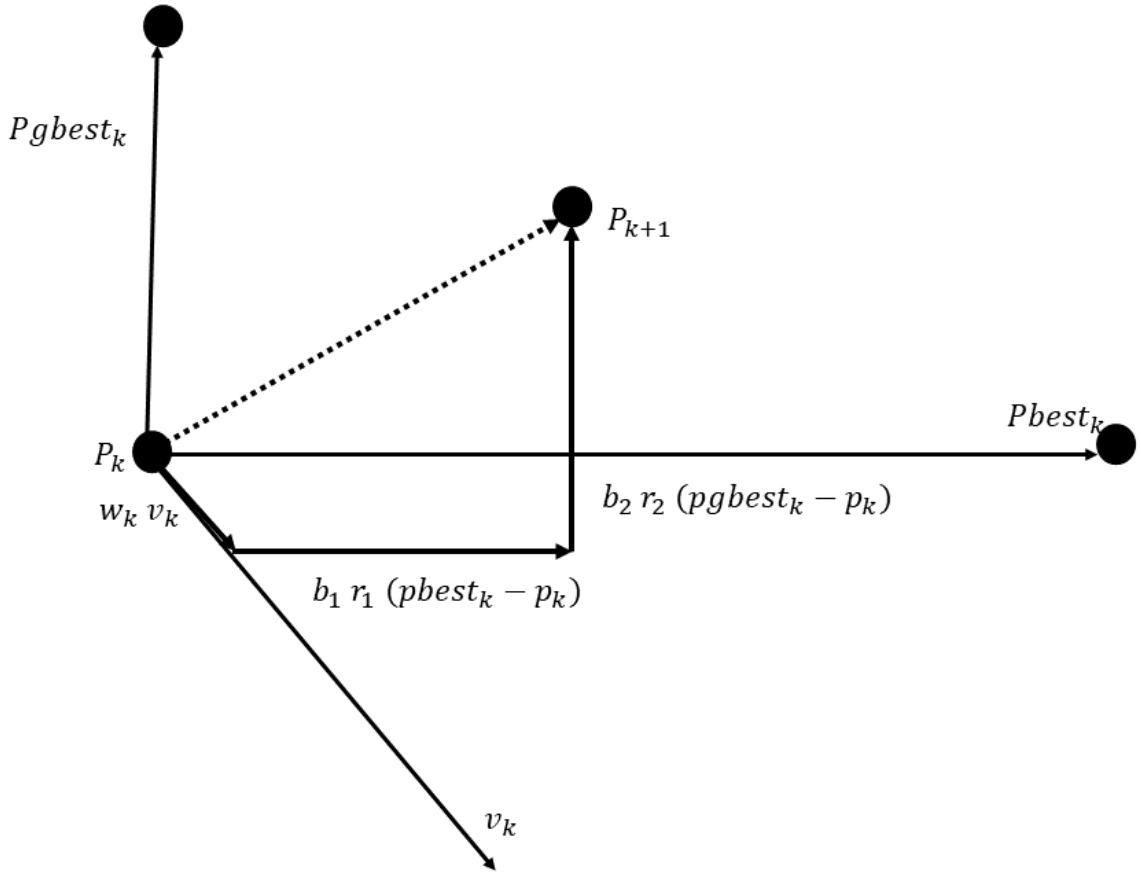

Fig. 7. Representation of the flight of a particle.

particle

$$\begin{pmatrix} & x_1 & x_2 & x_3 \\ particle_1 & 8.8 & 9.9 & 1.1 \\ particle_2 & 9.9 & 6.6 & 1.1 \\ particle_3 & 3.3 & 5.5 & 1.1 \\ particle_4 & 1.1 & 2.2 & 1.1 \\ particle_5 & 11 & 5.5 & 8.8 \end{pmatrix}$$

– We initialize the position (ϑ_i) randomly for each particle. We will consider:$\vartheta_i = 0.1x_i$; Then, the initial velocity of the first particle will be:

$$\begin{cases} \vartheta(x_1) = 0.1 * 8.8 = 0.88 \\ \vartheta(x_2) = 0.1 * 9.9 = 0.99 \\ \vartheta(x_3) = 0.1 * 1.1 = 0.99 \end{cases}$$

– The remaining particles behaved similarly as well, and finally we discover:

$$\begin{pmatrix} & x_1 & x_2 & x_3 \\ particle_1 & 0.88 & 0.99 & 0.11 \\ particle_2 & 0.99 & 0.66 & 0.11 \\ particle_3 & 0.33 & 0.55 & 0.11 \\ particle_4 & 1.1 & 0.22 & 1.1 \\ particle_5 & 1.1 & 0.55 & 0.88 \end{pmatrix}$$

2) Evaluation of the fitness function $f(x_i)$: Let's calculate the fitness value for each particle ; For example, its initial value for the first particle will be:

$$\begin{cases} \vartheta(x_1) & = 10(x_1 - 1)^2 + 20(x_2 - 2)^2 + 30(x_3 - 3)^2 \\ & = 10(8.8 - 1)^2 + 20(9.9 - 2)^2 + 30(1.1 - 3)^2 \\ & = 1.9649 * 10^{-3} \end{cases}$$

Next, we find the other fitness values for the other particles:

$$\begin{pmatrix} & \text{The objective function value for k = O} \\ particle_1 & 1.9649 * 10^3 \\ particle_2 & 1.3236 * 10^3 \\ particle_3 & 2.2179 * 10^3 \\ particle_4 & 2.9208 * 10^3 \\ particle_5 & 2.2542 * 10^3 \end{pmatrix}$$

Finally, we choose the best fitness value as g_best; in this case, we take $g_best = 1.3236 * 10^3$ and then $x_{g_{best}} = [x_1 = 9.9; x_2 = 6.6; x_3 = 1.1]$. For xbest of each particle, we take the position that gives the best value of gbest for each particle, and thus:

$$\begin{pmatrix} & x_{best} \\ particle_1 & [x_1 = 8.8; x_2 = 9.9; x_3 = 1.1] \\ particle_2 & [x_1 = 9.9; x_2 = 6.6; x_3 = 1.1] \\ particle_3 & [x_1 = 3.3; x_2 = 5.5; x_3 = 11] \\ particle_4 & [x_1 = 11; x_2 = 2.2; x_3 = 11] \\ particle_5 & [x_1 = 11; x_2 = 5.5; x_3 = 8.8] \end{pmatrix}$$

3) Update velocity and position for each particle: Let's calculate the position of the particles by:

$$x_{k+1} = x_k + \vartheta(k+1);$$

after computing the velocity by:

$$\vartheta_{k+1} = w.\vartheta_k + c_1.r_1.(x_{best_k} - x_k) + c_2.r_2.(x_{gbest_k} - x_k);$$

For the first particle in the initial iteration (k = 0), for instance:

$$\begin{cases} \vartheta_{k+1} & = w.\vartheta_k + c_1.r_1.(x_{best_k} - x_k) + \\ & \quad c_2.r_2.(x_{gbest_k} - x_k) \\ \vartheta_{0+1}(x_1) & = (0.9 * 0.88) + 0 + 2 * r_2.(9.9 - 8.8) \\ & = 1.0667 \\ \vartheta_{0+1}(x_2) & = (0.9 * 0.99) + 0 + 2 * r_2.(9.9 - 9.9) \\ & = -4.471 \\ \vartheta_{0+1}(x_3) & = (0.9 * 0.11) + 0 + 2 * r_2.(9.9 - 1.1) \\ & = 0.9000 \end{cases}$$

Then, we look for the values of the same variables for the other particles for the first iteration (k = 0), and we find:

- Velocity:

$$\begin{pmatrix} & \vartheta(x_1) & \vartheta(x_2) & \vartheta(x_3) \\ particle_1 & 1.066 & -4.4719 & 0.9 \\ particle_2 & 0.81 & 0.5400 & 0.09 \\ particle_3 & 7.4888 & 2.5728 & -18.3177 \\ particle_4 & -0.1678 & 1.4286 & 1.4286 \\ particle_5 & -1.2109 & 0.5286 & -13.66 \end{pmatrix}$$

- Next, position:

$$\begin{pmatrix} & \vartheta(x_1) & \vartheta(x_2) & \vartheta(x_3) \\ particle_1 & 9.8667 & 5.4281 & 2 \\ particle_2 & 10.71 & 7.14 & 1.19 \\ particle_3 & 10.78 & 8.07 & -7.317 \\ particle_4 & 10.83 & 3.628 & 12.42 \\ particle_5 & 9.78 & 6.028 & -4.8635 \end{pmatrix}$$

4) Evaluation of the fitness function $F(x_i)$: for the new values of velocity and position. For example, the new fitness value for the first particle at the new position:

$$\begin{cases} F(x_1) & = 10(x_1 - 1)^2 + 20(x_2 - 2)^2 + 30(x_3 - 3)^2 \\ & = 10(9.8667 - 1)^2 + 20(5.4281 - 2)^2 + 30 \\ & = 1.0512 * 10^{-3} \end{cases}$$

Then, we find the other fitness values for the other particles:

$$\begin{pmatrix} & \text{The value of the objective function} \\ particle_1 & 1.0512 * 10^3 \\ particle_2 & 1.5695 * 10^3 \\ particle_3 & 2.8866 * 10^3 \\ particle_4 & 2.6716 * 10^3 \\ particle_5 & 2.9504 * 10^3 \end{pmatrix}$$

The smallest value is then selected as the new fitness value after a comparison between the previous best fitness value and these new fitness values. So in this instance:

$$g_{best} = 1.0512 * 10^3;$$

$$\text{and } x_{gbest} = [x_1 = 9.8667; x_2 = 5.4281; x_3 = 2].$$

We compare its fitness value with the new one for each particle, and save those with the smallest value. We find:

$$\begin{pmatrix} & x_{best} \\ particle_1 & [x_1 = 9.8667; x_2 = 5.4281; x_3 = 2] \\ particle_2 & [x_1 = 9.9; x_2 = 6.6; x_3 = 1.1] \\ particle_3 & [x_1 = 3.3; x_2 = 5.5; x_3 = 11] \\ particle_4 & [x_1 = 11; x_2 = 2.2; x_3 = 11] \\ particle_5 & [x_1 = 11; x_2 = 5.5; x_3 = 8.8] \end{pmatrix}$$

5) Iteration update: Up till the stopping condition is satisfied, we go back and forth from step two to this one (i.e., reaching 50 iterations) $k = k + 1 = 0 + 1 = 1$.

6) Output with best g_{best} and x_{gbest} : Finally, the results obtained after 50 iterations: $g_{best} = 0.0032013791931731245$ and $x_{gbest} = [x_1 = 1; x_2 = 1.9874; x_3 = 3.0009888]$. The fourth particle is the one that provides us with this ideal answer. The graph in Figure. 8 depicts how the value of the objective function (g_{best}) has changed over time as a function of the number of iterations (k).

IV. CASE STUDY: OPTIMIZATION OF ANN LEARNING BY PSO

An ANN is made up of the association of simple objects known as neurons in a graph of varying complexity. The organization of the graph, or its architecture, the number of neurons, the presence or absence of feedback loops in the network, the type of neurons (their transition or activation functions), and finally the objective supervised or unsupervised learning, optimization, or dynamic systems distinguish the main networks. In this paper, we apply the PSO meta heuristics to the optimization problem of supervised learning of ANN using DRL.

A. ANN Learning process

For ANNs, learning entails computing the parameters so that the neural network's outputs are, for the examples used during training, as similar to the "desired" outputs as possible. The goal of ANN learning approaches, which are optimization algorithms, is to reduce the discrepancy between the network's actual answers and its desired replies by gradually changing its parameters (called "iterations").

As the learning process progresses, the output of the ANN matches the data ever-better. However, the ANN's error at the conclusion of learning is not zero. As a result of the noise that affects the measurements, we do not attempt to make the curve pass through each and every measurement point, nor do we attempt to replicate the measurement noise; instead, we aim to ensure that the error committed in the neural network's approximation is of the same order of magnitude as the noise that affects the measurements.

B. Optimization of ANN Learning Process using PSO

PSO algorithms are a recently proposed class of meta heuristics for optimization problems. In the next of this section we explain how we adapted PSO to ANN learning optimisation on different steps.

1) Adaptation of the PSO meta-heuristic to ANN: Think about a neural network with a single hidden layer and nc nodes that has a single input and a single output. These characteristics of this network:

- One input x and one output y ;
- Two weight vectors:
  - $W_1 = (w_{11}, w_{12}, ..., w_{1nc})$;
  - $W_2 = (w_{21}, w_{22}, ..., w_{2nc})$;

  withe w_{1i} : weight between the input and the intermediate layer and w_{2i}: weight between the intermediate layer and the output.

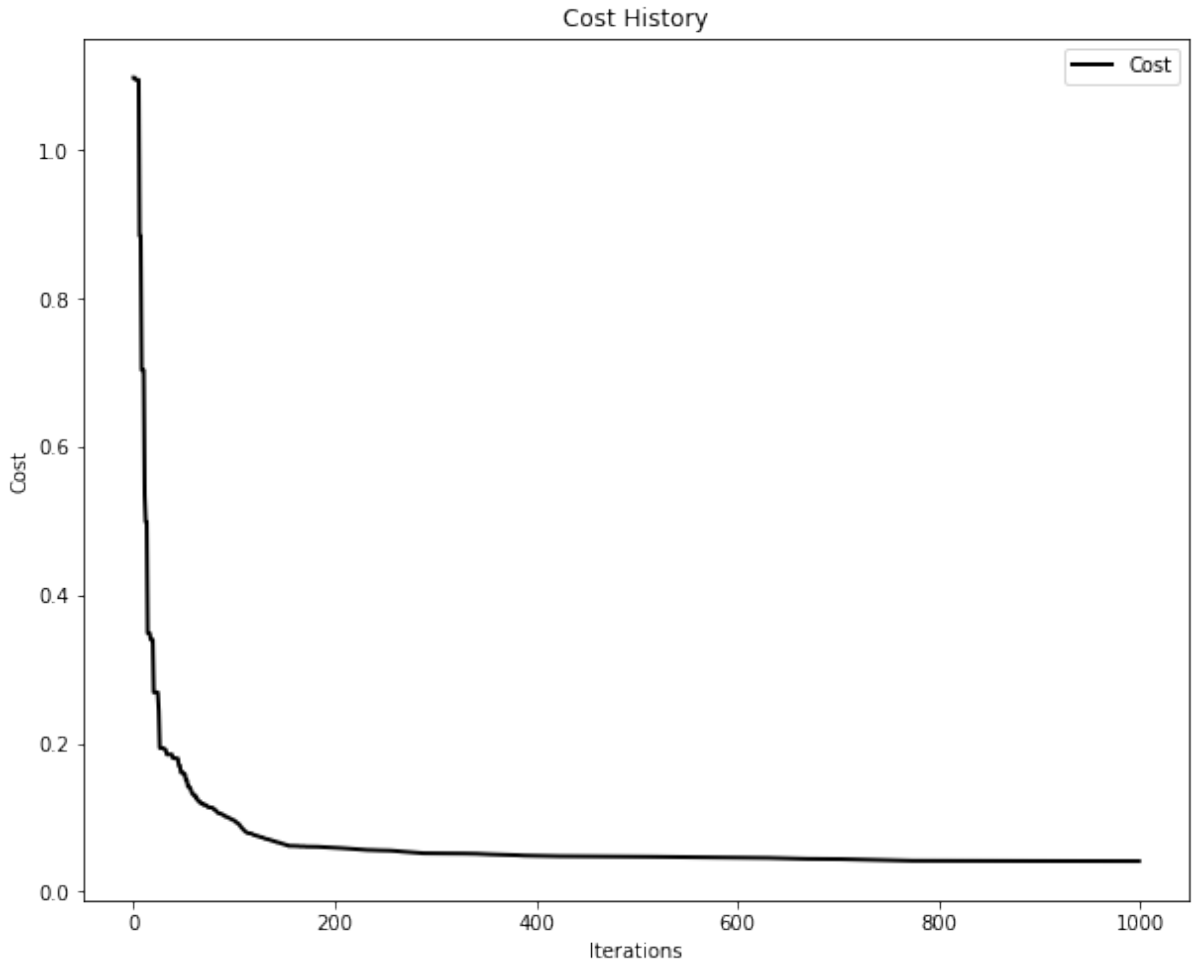

Fig. 8. The value of the objective function.

- A sigmoid function f at the hidden layer nodes;
- A linear function g at the node of the output layer;

The expression for the output $y(k)$ of this ANN network is:

$$Y(k) = \sum_{i=1}^{i=nc} w_{2i} * f(x(k) * w_{1i});$$

The following analogy needs to be used in order to apply the procedure to an ANN:

- A ANN represents a particle.
- A weight vector with the dimensions $(2 * nc)$ replaces the particle's position $x(k) : W = (W_1; W_2)$.
- A weight vector W takes the place of the particle's velocity or orientation.

$$W = (w_{v1}, w_{v2}, ..., w_{vnd}); nd = 2 * nc;$$

- The fitness function to maximize:

$$h(k) = 11 + e(k) \text{ With} : e(k) = |yr(k) - y(k)|$$

- In the discrete case $F(x_i) = F(x(k))$, the fitness function F calculates the position of the particle k, which is denoted by the expression $h(W)$: The fitness function value as a function of the weight vector denoting the particle's ANN.
- x_{pbesti}: The best position through which the particle i has passed. It is replaced by $Wpbest(k)$, the weight vector of the particle k which allowed the best fitness (maximum value of$h(k)$).
- $vpbesti$: The best velocity (orientation) that the particle has. It is replaced by $Wvbest(k)$ the weight vector of the particle k that allowed the best fitness $h(k)$ (maximum value).

C. PSO Optimization Algorithm

The method is to model DRL for reinforcement learning and instantiate a random number of ANNs (the number of particles for the PSO algorithm). The DRL consistently suggests changing the PSO algorithm's particle count in accordance with learning rate, which stands for reward.

D. ANN Training for Flower Classification "Iris" Data Set

Our Model is composed of 3 layers:

- An input layer with 4 neurons;
- An output layer with 3 neurons;
- An intermediate layer with 20 neurons;

And our optimization model contains 100 particles. Each particle position represents the weights of the neural network. The implementation steps of the algorithm are as follows:

- Import the libraries and dependencies necessary for the implementation of the model.
- Import the r2_score metric to evaluate the model.

Algorithm 2 Proposed Optimization Algorithm using PSO and DRL

repeat
  for k from $1tonc$ **do**
    if $h(\overrightarrow{x_i}) > p_{best_i}$ **then**
      $p_{best_k} \leftarrow h(\overrightarrow{x_i})$
      $\overrightarrow{w_{pbest_i}} \leftarrow w_k$
    end if
    if $h(\overrightarrow{(}k) > g_{best_i}$ **then**
      $g_{best_i} \leftarrow F(\overrightarrow{x_i})$
      $\overrightarrow{x_{gbest}} \leftarrow \overrightarrow{x_i}$
    end if
  end for
  for $ifrom1toN$ **do**
    $\overrightarrow{w_{pbest_i}} \leftarrow k(\overrightarrow{w_{pbest_i}} + p_1(\overrightarrow{w_{pbest_i}} - \overrightarrow{W_k})) + p_2(\overrightarrow{w_{vbest_i}} - \overrightarrow{W_k})$
    $\overrightarrow{W_k}) \leftarrow \overrightarrow{W_k} + \overrightarrow{w_{vbest_i}}$
  end for
  $iteration = iteration + 1$
  if $iteration = CONST$ **then**
    $iteration = 0$
    $reword = calculatethePSOconvergencerate$
    $Cr = DQRL(State, Reword)$
    if $Cr > 0$ **then**
      Reduce the number of particles furthest away;
    else
      Increase the number of particles around the best positions;
    end if
  end if
until stopping conditions

- Import "PySwarms".
- Import of the "Iris" data set.
- Store the features as X (inputs) and the labels as Y (outputs).
- Divide the data set into training and testing$(70\%, 30\%)$.
- Build the ANN model $[inputs = 4; n\_hidden = 20; n\_classes = 3]$.
- This implies the forward propagation of a single particle as an objective function to calculate the ANN forward propagation as well as the loss.
- The forward propagation must compute the backward propagation of weights and biases.
- Implement the forward propagation function for all particles as a higher level method to do forward-prop throughout the swarm.
- Initialize the swarm parameters with a dictionary whose keys contain the value of the specific optimization technique (cognitive, social, and inertia parameters).
- Perform the optimization to evaluate the objective function.

E. Optimization Results

The Figure. 9 represents the evolution of the value of the objective function (the cost function) as a function of the number of iterations (1000 iterations). The animation in Figure. 10 displays how two particle positions behave in relation to iterations:

We finally found the ideal neural network coefficients thanks to this optimization model and 1000 iterations.

V. Comparison of PSO to other meta-heuristic methods

We must first examine the concept of complexity in order to comprehend the significance of heuristics and meta-heuristics. The resources needed for an algorithm to function are represented as complexity. The calculation time is typically measured. P and NP are two distinct types of problems that make up the complexity. Problems that can be resolved in polynomial time are included in the P class. A problem is said to be in the P class if it can be solved effectively; otherwise, it is said to be tough. The solutions to problems in the class NP can be proven to be attainable in polynomial time. The problem of P = NP has not been resolved up to this point. If P is included in NP, the converse is not guaranteed. We opted to compare the PSO with the Genetic Algorithm (GA) and the Firefly Method as two additional meta-heuristic search methods for this study (Firefly).

A. Comparison between PSO and GA

A relatively new heuristic research technique called particle swarm optimization (PSO) draws its mechanics from the swarming or cooperative behavior of biological populations. PSO and the genetic algorithm (GA) are both population-based search techniques, making them both evolutionary heuristics. In other words, PSO and GA use a combination of deterministic and probabilistic rules to go from one set of points (population) to another set of points in a single iteration with a likely improvement.

Because of their simplicity, ease of use, and capacity to effectively resolve highly nonlinear mixed integer optimization problems typical of complex engineering systems, GA and its various variations have gained popularity in academia and industry. GA's drawback is its high computational expense. In terms of identifying the real global optimal solution, PSO is just as effective as GA when statistical analysis and formal hypothesis testing are used, but with much greater computational efficiency (fewer function evaluations) than GA. A collection of benchmark test issues and two space system design optimization problems, telescope array configuration and spacecraft reliability-based design, are used to compare the performances of GA and PSO.

B. Comparison of PSO and Firefly

The outcomes in Table I are provisional outcomes. Further investigation and measurement are required because there can still be mistakes and inaccuracies.

We shall express a view on the application of optimization methods based on previous literature investigations. This is because PSO might offer many different solutions while having no guiding principles for the problem that has to be optimized. PSO and other meta heuristic techniques cannot guarantee the best outcomes. More specifically, PSO differs from traditional optimization techniques like quasi newton and derivative gradient in that it does not call for the optimization of gradient issues. The PSO method and the EC (Evolutionary Computation) method are quite similar in many ways.

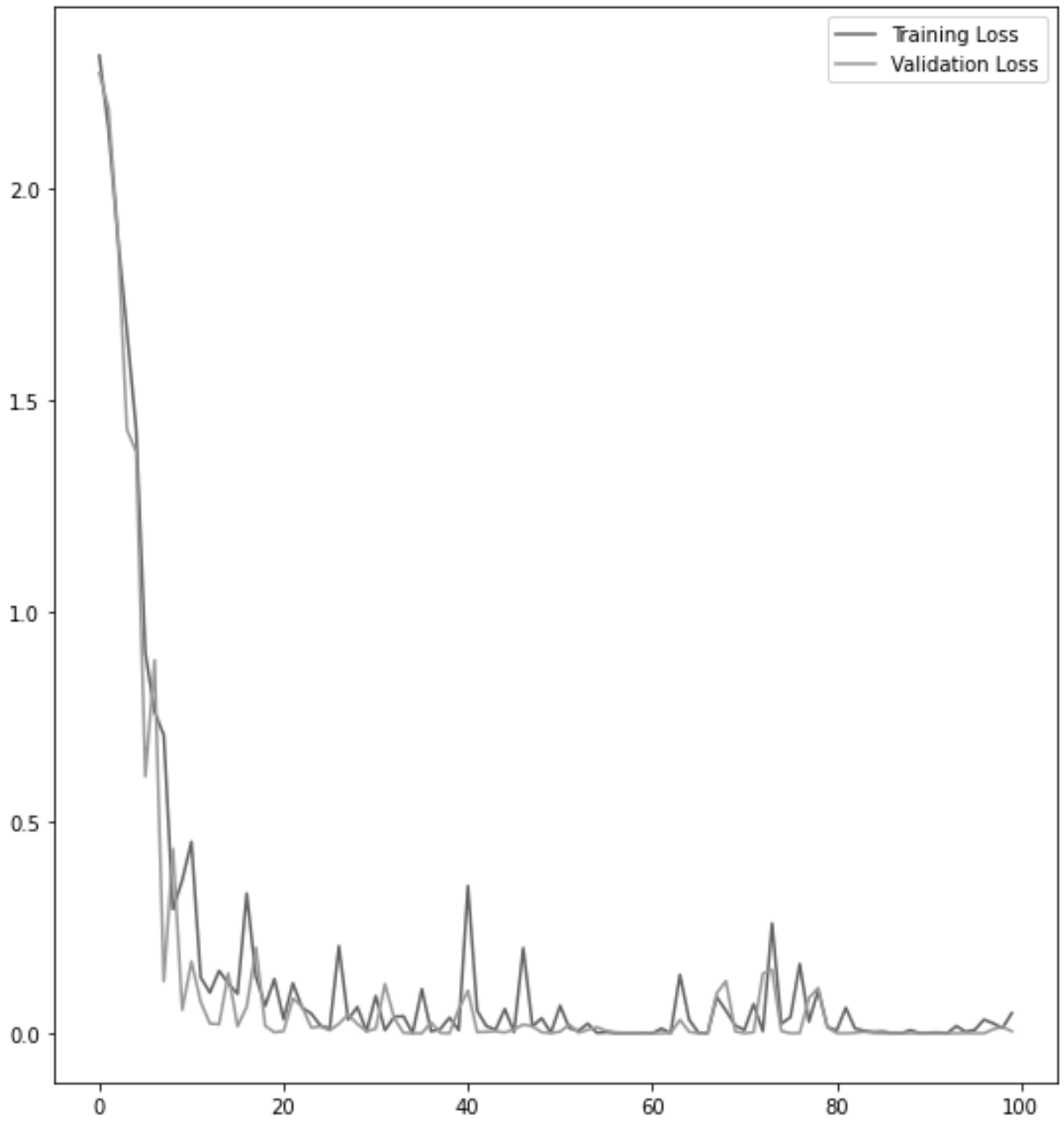

Fig. 9. The value of the objective function.

TABLE I
PSO AND FIREFLY PERFORMANCE COMPARISON

Feature	PSO	FireflyOne
Metaheuristic	⊗	⊗
Flexibility	⊗	⊗
Genetic operators		⊗
Low time complexity	⊗	⊗
Easy to modify	⊗	
Use of randomness		⊗
Certain of the most optimal solutions		⊗
Convergence	⊗	⊗

Both methods begin with a collection of randomly produced populations and assess the population as a whole using fitness values. The fundamental distinction between PSO and other optimization techniques is that PSO does not use genetic operators like crossover or mutation.

The PSO method's particles use internal velocity to update the data. As many times as necessary, the updating procedure is repeated. Only the best particles will be used to create the ideal solution in the final iteration. Implementation is simple due to the lack of genetic operators like cross-SOP clarity methods. It is now used as a vocational technique. Because the PSO method uses very few parameters and has a low time complexity, we may conclude that it is a straightforward method.

There are numerous meta heuristic techniques, and each has benefits and drawbacks of its own. PSO variants, GA variants, and other variants were created as a result of some researchers' attempts to enhance things, which had their ups and downs. The alternatives generated on one side are quicker, less complex, but have a less ideal answer, while the opposite is true.

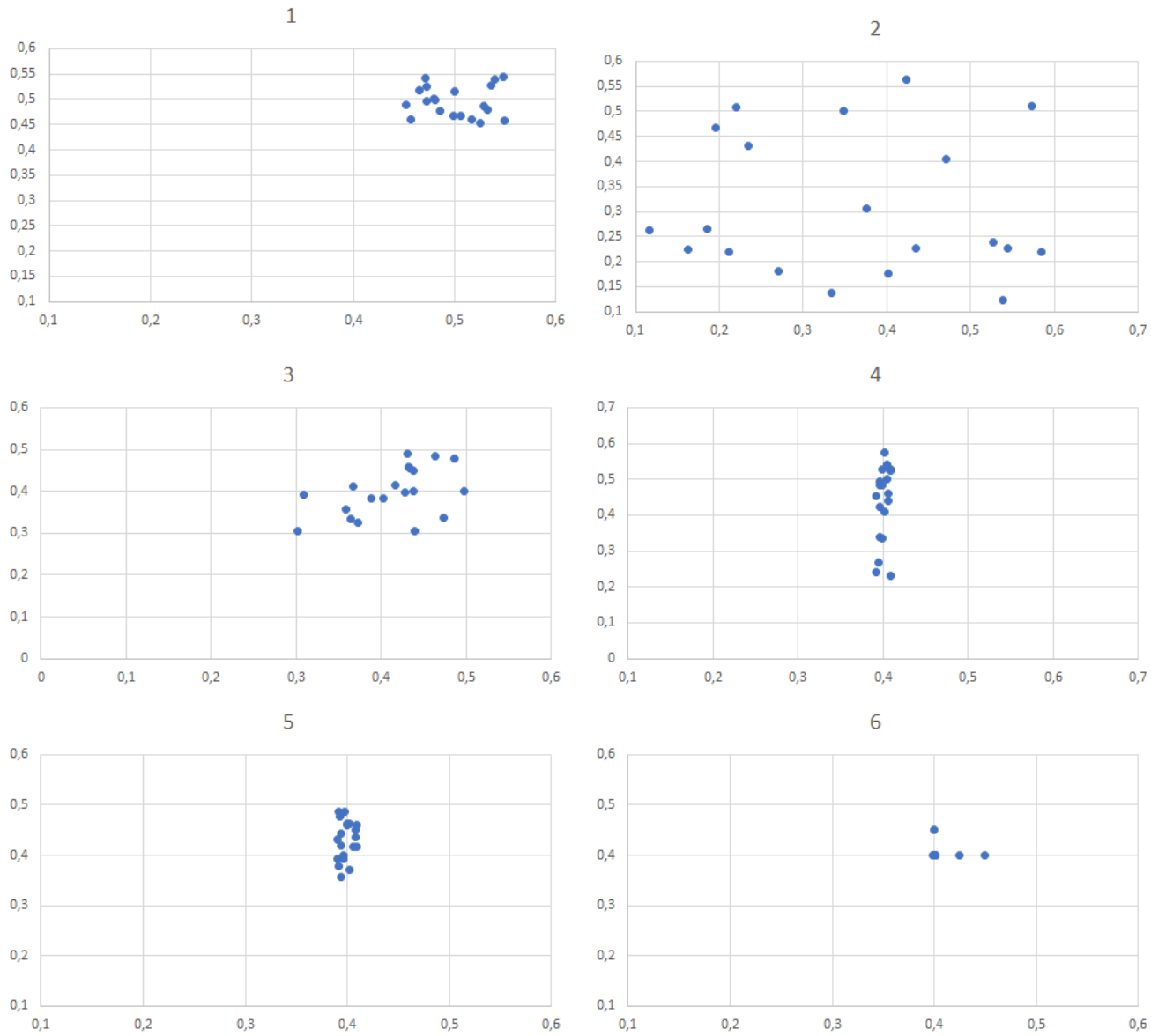

Fig. 10. Convergence of PSO particles towards the optimal solution: from initial state (1) to final state (6).

The convergence of population diversity loss is one serious issue that the PSO approach heavily depends on. There are many more frequent applications for global optimization techniques. other issues with simple solutions. The firefly algorithm is another another meta-heuristic technique. The firefly approach relies on chance to identify answers to issues. Even with the same characteristics, not all problems can be solved by the same solution. Directly observable natural occurrences serve as the basis for the Firefly technique. Whereas the masculinity-based approach is ineffective, the condition can find the best treatment with this normal approach.

We are aware of the method for utilizing each optimization algorithm thanks to the explanation in the previous section. When selecting an algorithm, researchers must take care to consider the data and the degree of difficulty they will face. Another item to keep in mind is the goal of the research, namely the importance of the desired optimization and how it will affect the study's conclusions. Here in Table II, are a few advantages realized by utilizing the previously mentioned optimization algorithm.

TABLE II
THE ADVANTAGES OF USING THE PSO TO FIREFLY.

Algorithm	Advantages
PSO	Easy to implement Requires only a few parameters There is no evalution or mutation in the operator Requires less computing More flexible in maintaining
Firefly Algorithm	Efficient to solve complex problems Low time complexity Can be used for various optimization problems

VI. CONCLUSION

The particle swarm optimization (PSO) algorithm was introduced in this study and was inspired by the world of animals (bird species). Since its introduction, this approach has had great success since it is straightforward and works

well for a variety of issues without forcing the user to change the algorithm's fundamental structure. Although the PSO has experienced substantial issues (such as premature convergence, which can cause algorithms of this type to become stuck in a local optimum, for example), it has been able to be significantly and permanently improved because to its capacity and evolutionary nature.

The task involved modifying the PSO meta-heuristic for neural network optimization, specifically to identify the weight vector that would enable us to get the greatest fitness function value.

It would be simple to apply the PSO to ANNs. However, it is suggested that simple and reduced structures be used, together with a sufficient number of particles in relation to the size of the search space and the difficulty of the optimization task.

Particularly to the MLP structure, the PSO has been well adapted to neural networks. Contrary to precise approaches, using PSO does not necessitate that the objective function being optimized is derivable or continuous. To reduce processing time, it is always preferable to utilize basic structures with a single hidden layer and fewer nodes and connections. Then, one should search for a balance between the number of particles (neural networks) and algorithm performance.

REFERENCES

[1] I. E. Hassani, C. E. Mazgualdi, and T. Masrour, "Artificial intelligence and machine learning to predict and improve efficiency in manufacturing industry," *arXiv preprint arXiv:1901.02256*, 2019.

[2] C. El Mazgualdi, T. Masrour, I. El Hassani, and A. Khdoudi, "Machine learning for kpis prediction: a case study of the overall equipment effectiveness within the automotive industry," *Soft Computing*, vol. 25, no. 4, pp. 2891–2909, 2021.

[3] R. L. I. E. T. M. Tarik, Hajji, "Optimizations of distributed computing processes on apache spark platform," *IAENG International Journal of Computer Science*, vol. 50, no. 2, pp. 422–433, 2023.

[4] M. T. O. J. M. I. Z. F. S. . J. E. Hajji, T., "Distributed and embedded system to control traffic collision based on artificial intelligence," *In Artificial Intelligence and Industrial Applications: Smart Operation Management*, pp. 173–183, 2021.

[5] J. Li, J.-h. Cheng, J.-y. Shi, and F. Huang, "Brief introduction of back propagation (bp) neural network algorithm and its improvement," in *Advances in Computer Science and Information Engineering*. Springer, 2012, pp. 553–558.

[6] M. O. Tarik, H., "Big data analytics and artificial intelligence serving agriculture." *In Advanced Intelligent Systems for Sustainable Development*, pp. 57–65, 2020.

[7] N. Ouerdi, T. Hajji, A. Palisse, J.-L. Lanet, and A. Azizi, "Classification of ransomware based on artificial neural networks," in *International Conference Europe Middle East & North Africa Information Systems and Technologies to Support Learning*. Springer, 2018, pp. 384–392.

[8] J. O. Tarik, H., "Weather data for the prevention of agricultural production with convolutional neural networks." *International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, pp. 1–6, 2019.

[9] E. J. S. Y. M. J. . J. E. M. Hajji, T., "Microfinance risk analysis using the business intelligence." *International Colloquium on Information Science and Technology (CiSt)*, pp. 675–680, 2016.

[10] M. K. Tarik, Hajji and J. E. Miloud., "Digital movements images restoring by artificial neural netwoks." *Computer Science and Engineering*, pp. 36–42, 2014.

[11] H. Miyajima, N. Shigei, H. Miyajima, and N. Shiratori, "Securely distributed computation with divided data and parameters for hybrid particle swarm optimization," *IAENG International Journal of Applied Mathematics*, vol. 52, no. 3, pp. 541–549, 2022.

[12] P. Pedregal, *"Introduction to optimization"*. Springer, 2004, vol. 46.

[13] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[14] R. V. Rao and V. J. Savsani, "Mechanical design optimization using advanced optimization techniques," 2012.

[15] X. Ding, X. Liu, and H. Li, "Improved branch and bound global optimization algorithm for a class of sum of linear ratios problems," *IAENG International Journal of Applied Mathematics*, vol. 52, no. 3, pp. 617–624, 2022.

[16] I. H. Osman and J. P. Kelly, "Meta-heuristics theory and applications," *Journal of the Operational Research Society*, vol. 48, no. 6, pp. 657–657, 1997.

[17] F. W. Glover and G. A. Kochenberger, *"Handbook of metaheuristics"*. Springer Science & Business Media, 2006, vol. 57.

[18] J. Qian and G. Chen, "Improved multi-goal particle swarm optimization algorithm and multi-output bp network for optimal operation of power system," *IAENG International Journal of Applied Mathematics*, vol. 52, no. 3, pp. 576–588, 2022.

[19] T. Hajji, A. A. Hassani, and M. O. Jamil, "Incidents prediction in road junctions using artificial neural networks," in *IOP Conference Series: Materials Science and Engineering*, vol. 353, no. 1. IOP Publishing, 2018, p. 012017.

[20] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[21] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *"Complexity and approximation: Combinatorial optimization problems and their approximability properties"*. Springer Science & Business Media, 2012.

[22] K.-L. Du, M. Swamy *et al.*, "Search and optimization by metaheuristics," *Techniques and Algorithms Inspired by Nature*, pp. 1–10, 2016.

[23] Y. Zhang, S. Wang, and G. Ji, "A comprehensive survey on particle swarm optimization algorithm and its applications," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[24] S. Desale, A. Rasool, S. Andhale, and P. Rane, "Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey," *Int. J. Comput. Eng. Res. Trends*, vol. 351, no. 5, pp. 2349–7084, 2015.

[25] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4. Citeseer, 1995, pp. 1942–1948.

[26] D. Wolpert, "Theory of collective intelligence," in *Collectives and the Design of Complex Systems*. Springer, 2004, pp. 43–106.

[27] A. Gloria, S. Neukamm, and F. Otto, "A regularity theory for random elliptic operators," *arXiv preprint arXiv:1409.2678*, 2014.

[28] L. J. V. Miranda, "Pyswarms documentation," 2020.