IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 15, 2018, accepted November 7, 2018, date of publication November 29, 2018,
date of current version January 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2882568

Raccoon Optimization Algorithm

SINA ZANGBARI KOOHI[1], NOR ASILAH WATI ABDUL HAMID[1], MOHAMED OTHMAN[1], AND GAFURJAN IBRAGIMOV[2]

[1]Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia, Seri Kembangan 43400, Malaysia

[2]Department of Mathematics, Faculty of Science, University Putra Malaysia, Seri Kembangan 43400, Malaysia

Corresponding author: Nor Asilah Wati Abdul Hamid (asila@upm.edu.my)

This work was supported by Universiti Putra Malaysia under Grant GP/2017/9569600.

ABSTRACT Population-based meta-heuristic is a high-level method intended to provide sufficient solution for problems with incomplete information among a massive volume of solutions. However, it does not guarantee to attain global optimum in a reasonable time. To improve the time and accuracy of the coverage in the population-based meta-heuristic, this paper presents a novel algorithm called the raccoon optimization algorithm (ROA). The ROA is inspired by the rummaging behaviors of real raccoons for food. Raccoons are successful animals because of their extraordinarily sensitive and dexterous paws and their ability to find solutions for foods and remember them for up to three years. These capabilities make raccoons expert problem solvers and allow them to purposefully seek optimum solutions. These behaviors exploited in the ROA to search the solution spaces of nonlinear continuous problems to find the global optimum with higher accuracy and lower time coverage. To evaluate the ROA's ability in addressing complicated problems, it has been tested on several benchmark functions. The ROA is then compared with nine other well-known optimization algorithms. These experiments show that the ROA achieves higher accuracy with lower coverage time.

INDEX TERMS Raccoon optimization algorithm (ROA), nonlinear continuous optimization problems, structural optimization, evolutionary algorithm, meta-heuristic algorithm.

NOMENCLATURE

G_{opt}	Global Optimum
loc_i	Location of the raccoon in current iteration
MF	Migration Factor
$n_{preserve}$	Perseveration Parameter
$N_{reachable}$	Number of reachable zone candidates
$N_{visible}$	Number of visible zone candidates
NI	Number of iterations
R_{best_i}	Best solution among reachable zone population in current iteration
ROA	Raccoon Optimization Algorithm
RZP	Reachable Zone Population
RZR	Reachable Zone Radius
V_{best_i}	Best solution among visible zone population in current iteration
VZP	Visible Zone Population
VZR	Visible Zone Radius

I. INTRODUCTION

Developing algorithms is a well-known and important aspect of mathematics and computer science. One of the special cases of developing algorithms is nature-inspired algorithms. They principally inspired by two basic ideas:

- The human brain;
- The evolutionary process.

Algorithms based on the evolutionary process are a successful and well-studied category of natural algorithms. They are computational methods inspired by the natural evolutionary process. The relation between natural evolution and problem solving is illustrated in Figure 1a [1].

Evolutionary algorithms fall into the category of meta-heuristics algorithms and are widely used for problems such as NP-hard combinatorial problems [2], job scheduling problems [3], and multi-objective optimization problems [4]–[10]. Meta-heuristic algorithms have two main directions: intensification and diversification. Intensification refers to searching near the current best solutions (local optima), while diversification refers to searching widely for better solutions (global optima) [11]. The other promising directions of meta-heuristic algorithms are having many neighborhood searches to avoid of getting stuck in local optimums [12], providing constructive basic to increase the quality of heuristic selection and consequently increase the accuracy, reducing

the iterations as much as possible to have shorter coverage time [13].

By a quick search on *Wikipedia* a numerous state-of-the-art meta-heuristics and evolutionary algorithms can be found. Some of the most recent algorithms are Hydrological cycle Algorithm [14], Spiral optimization Algorithm [15], Invasive Weeds Optimization (IWO) [16], Killer Whale Algorithm [17], An Improved Teaching-Learning-Based Optimization [18], Gray Wolf optimization [19], and Fruit Fly Optimization Algorithm [20]. Some of these algorithms will be discussed in detail in the next section. It has been demonstrated that the aforementioned algorithms solve or improve the optimum solution for various kinds of problems. But, there are two common and general questions ''which algorithm is better than the others?'' and ''by having the tsunami of novel meta-heuristics algorithms'', is there still need to propose a new algorithm?. As Sörensen [21] nicely explained, the answer to the first question is that ''*it depends*''. It depends on the problem domain and is possible to determine which algorithm is more suitable for a specific domain of optimization problem. For the second question, as meta-heuristics algorithms do not guarantee the most optimum solution within a reasonable time, where this field still needs to have novel algorithms. However, they must follow the aforementioned promising direction of meta-heuristics algorithms and have effective improve on it.

In this study, a novel algorithm named the Raccoon Optimization Algorithm (ROA) which is fully shaped based on the promising direction of the meta-heuristics optimization algorithm is proposed. ROA by inspiring from racoon's life cycle which is searching in two different zones simultaneously for food provides a lightweight technique to explore solution state space. This feature both reduces the risk of becoming stuck in local optima and saves time during exploration. Raccoons also are known as expert problem solvers that can remember problem solutions for up to three years; similarly, the ROA remembers its best solution and moves purposefully toward the optimum solution. Searching in the solution state space based on this technique leads to reduce the times of iteration and consequently the coverage time remarkably reduce. Moreover, the complexity of ROA is less than previous complex evolutionary algorithms such as artificial fish [22]. This makes ROA even faster than the others. It can be said that the overarching goal behind the ROA is to create a meta-heuristics algorithm that is able to reach the optimum solution in a shorter time. The problem domain that ROA is suited for is combinatorial problems.

The remainder of this paper is organized as follows. In Section II, a brief history of evolutionary algorithms is given, related research works in this area are studied, and their strengths and weaknesses are discussed. Section III provides a short description of the raccoon's natural lifestyle which inspired our proposed approach. In Section IV, the ROA is presented, and its implementation details are given. In Section V, the ROA is used to solve different benchmark functions, and its results are compared with those of similar algorithms. Finally, Section VI discusses and concludes this work and presents suggestions for future work.

II. BRIEF HISTORY OF EVOLUTIONARY ALGORITHMS AND RELATED WORKS

The application of Darwinian Principles (the natural process of evolution) [23] in solving scientific problems began in the 1940s before the emergence of computers [13]. In 1948, Turing proposed ''genetical or evolutionary search'' [24]. The first computer-based optimization experiments using these ideas were conducted in 1962 by Bremermann [25]. In the 1960s, three main implementations of this idea were developed that separated evolutionary algorithms into three main categories:

- **Evolutionary Programming**, proposed by Fogel *et al.* [26] in the USA;
- **Genetic Algorithm**, proposed by Holland [27] in the USA;
- **Evolution Strategies**, proposed by Rechenberg [28] and Schwefel [29] in Germany.

However, since the 1990s, the computer science research community has combined these three categories into a broader field called **Evolutionary Computing (EC)**, and each of the existing methods became a category in this field [30]–[32].

Without a doubt, the 1980s and 1990s were the age of efflorescence for meta-heuristic algorithms. In 1983, a new optimization technique named simulated annealing (SA) was proposed by Kirkpatrick *et al.* [33]. This algorithm called Simulating Annealing (SA) inspired by the annealing process of metals. SA explores solution state space sequentially, therefore, some of the candidate solutions may be revisited many times. It leads to an increase in computation time without any enhance upon optimum solution [34]. To tackle this problem, in 1986, Fred Glover introduced a new algorithm named Tabu search. This algorithm was probably the first in this field to make use of memory in meta-heuristics [35] by having a list called tabu list. Avoidance of revisiting is depended on the size of this list. A large tabu list also cannot be efficient enough and solve SA problem.

Dorigo [36] defended his PhD thesis in 1992 at the University of Politecnico di Milano; his dissertation introduced a new optimization algorithm named Ant Colony Optimization (ACO). ACO is more efficient for the problems like traveling salesman and vehicle routing [37]. An American social psychologist named James Kennedy and an American engineer named Russell C. Eberhart in 1995 jointly proposed the Particle Swarm Optimization (PSO) algorithm [38]. PSO has been studied, improved and utilized in numerous problems. The important issue in the PSO that should be considered is the local coverage. PSO is able to find local coverage of one-dimensional problems, but it has been proven that it cannot find local coverage when the dimension is more than one [39]. Moreover, there is no guarantee that the coverage point of a particle is a local optimum [40]. Later, in 1996,

Storn and Price [41] implemented an innovative algorithm named Differential Evolution (DE).

Up to that point, research had focused on the search for algorithms that obtained better results than previous algorithms. However, in 1997, Wolpert and Macready [42] shocked the evolutionary algorithm community by demonstrating that no universal best algorithm exists. Instead, each algorithm achieves good results for some types of optimization problems, but other algorithms will perform better on other types. It also has been nicely explained by Sörensen [21].

The search for optimization algorithms continues in the 21st century. It started with the Harmony Search (HS) algorithm, which was proposed by Geem *et al.* [43]. Despite its advantages, HS still needs further improvement on coverage time, accuracy, global optimum and having more user-friendly parameters-setting [44]. In 2004, Nakrani and Tovey [45] presented a new algorithm called the Honey Bee Algorithm (HBA). One year later, inspired by HBA, two other algorithms were presented: a Bee Algorithm (BA) by Pham *et al.* [46] and the Artificial Bee Colony (ABS) algorithm by Karaboga [47]. Then, many improvements for HBC like artificial bee colony algorithm have proposed in [48], the bees are using a solution update during searching for new sources of foods. The solution update strategy has some issues like slow coverage of optimal or even near-optimal solutions [49]–[51] and becoming stuck in the local optimum [50], [51].

Then another algorithm called Firefly Algorithm (FA) proposed by Yang [52]. Standard FA, like many other algorithms, needs improvement on local and global search [53], [54]. In 2009, two researchers from Cambridge University in the UK and Raman College of Engineering in India, named Xin-She Yang and Suash Deb, presented a new algorithm named the Cuckoo Search (CS) algorithm [55]. The coverage speed of the original CS is not efficient and has been improved slightly by some researchers like [56]. The original CS also does not support multi-objective problems [57]. It also needs improvement on searching global solution [58]. Yang [59] developed the Bat algorithm in 2010. The most important advantages of the BA is quick coverage that makes it proper when a fast algorithm is needed like classification. Even though BA is efficient but it still requires to improve parameter tuning, parameter control and accuracy [60]. Figure 1b shows a timeline of the history of optimization algorithms up to 2010.

From 2010 till now there are numerous algorithms have been proposed in the literature that are surveyed in many papers like [61]–[71]. Each of these algorithms have its own pros and cons, however, due to lack of space we only mention the common drawbacks between them: they mostly need great effort to utilize, they have complicated parts or details therefore it is hard to understand, inefficient number of iteration and lack of optimum coverage time.

Hence, the ROA that presented in this study attempts to gain improvements such as easy understanding and application, simple computation process, a small number of iterations, fast coverage time, and higher accuracy.

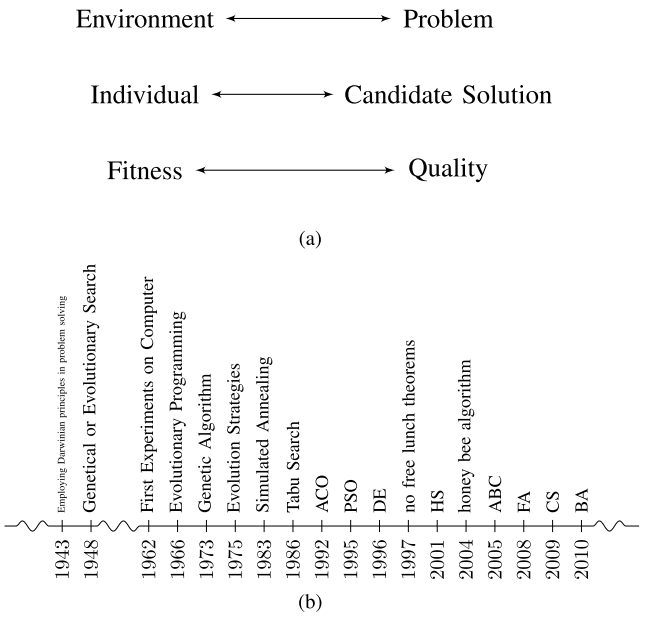

FIGURE 1. (a) Natural Evolution vs. Evolutionary Algorithms. (b) Brief History of the EA.

III. RACCOON LIFESTYLES

The raccoon (sometimes spelled racoon) is a mammal that originated in North America. Raccoons are known as intelligent and curious creatures. These qualities, combined with their dexterous paws, make raccoons extremely successful in searching for food; raccoons have often been known to make its way into houses to find food.

There is an old myth that raccoons wash their food in water before eating it. However, the truth is something beyond washing. The animal's paws have more touch receptors than any other part of its body, and these receptors are more sensitive when wet. Additionally, scientists showed that a large portion of the processing area of the raccoon's brain is dedicated to its paws. Thus, it dips its food into water to make its paws more sensitive and examine the food more carefully [72].

Raccoons are colorblind; consequently, they are poor at distinguishing colors. However, their eyes are very sensitive to green light, and they are able to see well at night and in twilight. Their eyes, in combination with their powerful sense of smell, are their second tool for finding food. Moreover, zoologists believe that raccoons have an excellent memory; they can remember events even three or four years later [72], [73].

The success of this animal in finding food in a short time and its good memory were the main inducements to proposing the ROA. Considering the solution domain of the problem as the living environment of the animal, the raccoon, using its amazing paws and vision, tries to find the best food (solution) in its realm.

IV. RACCOON OPTIMIZATION ALGORITHM (ROA)

The process of finding the global optimum for a predefined fitness function in the ROA is inspired by the natural food searching habits of raccoons. In this process, the domain of the fitness function is taken as the living environment of a raccoon. Food, which represents different possible solutions of the fitness function, is spread throughout the raccoon's living environment. The goal is to find the best solution among all the possible solutions. This algorithm makes use of two different population sets: the Reachable Zone Population (*RZP*) and the Visible Zone Population (*VZP*). These population sets will be explained in detail later.

The total process of the algorithm is divided into three steps: i) Parameter Definition, ii) Initialization and iii) Main Loop. Each step will be studied in detail in the following sections.

A. PARAMETER DEFINITION

This step defines various algorithm parameters. The parameters control the general behavior of the algorithm and should be set in a problem-specific manner. The parameters are as follows.

1) REACHABLE ZONE RADIUS

Unlike other optimization algorithms, in the ROA, the candidate solutions in each iteration cannot stem from anywhere inside the problem solution domain; instead, the candidate solutions in this population must be inside the reachable zone of the animal. Thus, we define a radius for this population, which is the maximum distance of this population's members from the current location of the animal, named *loc*. We call this radius the Reachable Zone Radius (*RZR*). The *RZR* is specified by the user.

2) REACHABLE ZONE CARDINALITY

The reachable zone cardinality represents the number of solution candidates inside the reachable zone population (*RZP*). The number of candidate solutions that *RZP* contains, is called $N_{reachable}$ and is specified by the user as a parameter. The *RZP* is discussed in detail in the next section.

3) VISIBLE ZONE RADIUS

The area in which the raccoon can check for food visually is called the visible zone. The size of this zone is limited by the Visible Zone Radius (*VZR*). The *VZR* is the maximum distance at which the raccoon can find the current food location and is specified by the user.

4) VISIBLE ZONE CARDINALITY

The visible zone cardinality represents the number of solution candidates inside the visible zone population (*VZP*). This value is denoted as $N_{visible}$ and is declared by the user as a parameter. However, because a raccoon's eyes are as not precise as its paws, the number of candidates it can test visually should be smaller than the number of candidates it can touch.

$$N_{visible} < N_{reachable} \tag{1}$$

Pseudocode 1 Step 1. Parameter Definition

```
RZR ← Reachable Zone Radius
N_reachable ← Number of Reachable Zone Candidates
VZR ← Visible Zone Radius
N_visible ← Number of Visible Zone Candidates
NI ← Number of Iterations
MF ← Migration Factor
```

5) NUMBER OF ITERATIONS

As is commonly the case for optimization algorithms, the ROA is repetitive in nature. Thus, it repeats its main process by a predefined number of times. The maximum number of iterations or repetitions is defined by the user using a parameter named *NI*.

6) MIGRATION FACTOR

Optimization algorithms can become stuck in local optima. This occurs when the algorithm finds a solution that is optimal in part of the solution domain even though better solutions might exist in other parts of the solution domain. This local optimum can deceive the algorithm and cause it to return a solution that is not globally optimal. The ROA simultaneously searches in two different zones, which reduces the risk of becoming stuck in a local optimum and improves the algorithm's global exploration ability. Nevertheless, becoming stuck in a local optimum might still occur. Thus, the ROA uses *migration*, which is a technique to help avoid local optimum traps. Migration occurs when the algorithm remains in a specific location for a predefined number of iterations. This condition is named *perseveration*. The predefined number of iterations at which perseveration occurs is called the Migration Factor (*MF*) and is specified by the user as a parameter. Perseveration and migration are discussed in more detail in the next section.

Pseudocode 1 shows a summary of the essential parameters for the algorithm.

B. INITIALIZATION

After setting the parameters of the algorithm, the next step is to initialize it. ROA initialization consists of three steps. The first step defines the initial location of the raccoon; the second step builds the initial reachable population; and the last step constructs the visible population. Each step will be described in detail in the following sections.

1) INITIAL LOCATION

First, a random location is assigned to the raccoon in the problem solution domain using a random function. In this algorithm, the location of the raccoon in iteration i, $i \in \{0, 1, \cdots, NI\}$, is denoted as loc_i. The initialization step is considered iteration zero; thus, the initial location is loc_0.

As mentioned in the previous chapter, the raccoon has a very good memory; hence, in this algorithm, it remembers the location of the best food that it has found in its whole life. This location is called the global optimum (G_{opt}). Initially, in iteration zero (initialization), because no location has been evaluated yet, G_{opt} is set to the current random location of the animal.

$$G_{opt} = loc_0 \tag{2}$$

2) INITIAL REACHABLE POPULATION

After setting loc_0, a reachable population is built around it. The first population of candidate solutions in the ROA is named the Reachable Zone Population (*RZP*). The *RZP* is defined by a circle around the raccoon with radius equal to the *RZR* and includes a set of possible solutions. This population represents the set of possible food items around the animal that it can reach with its paws. The set of possible solutions in this area might be enormous; therefore, only a random subset of the candidates is selected for examination:

$$RZP = \{r_0, r_1, \ldots, r_{N_{reachable}}\} \tag{3}$$

where $N_{reachable}$ is the number of reachable zone candidates, which is specified by the user, and $r_i, \forall i \in \{0, 1, \cdots, N_{reachable}\}$ are random candidates that indicate current food locations. The r_i in an N-dimensional optimization problem is defined as follows:

$$r_i = (x_0, x_1, \ldots, x_{N_{reachable}}), \quad \forall i \in \{0, 1, \cdots, N_{reachable}\} \tag{4}$$

Considering $\delta(loc, candidate)$ as the distance of the candidate solution from the raccoon's current location (loc), the candidates inside *RZP* must satisfy the following relation:

$$0 < \delta(loc, r_i) \leq RZR; \quad \forall i \in \{0, 1, \cdots, N_{reachable}\} \tag{5}$$

RZP is important in this algorithm because a raccoon's sense of touch is its most powerful sense, which it uses to examine the possible solutions more precisely. Precision here refers to the large number of solution candidates that are examined by the raccoon. Therefore, the initial reachable zone population, denoted by RZP_0, is defined as follows:

$$RZP_0 = \{r_i | i \in \{0, 1, \cdots, N_{reachable}\}, \; 0 < \delta(loc_0, r_i) \leq RZR\} \tag{6}$$

After forming the initial reachable population (RZP_0), the algorithm searches for the best solution among this group; this best solution is called R_{best_0}. Considering the optimization problem as a maximization problem and f as the fitness function to be optimized, we have

$$\begin{aligned} R_{best_0} &= r_j \\ &\text{where} \\ r_j &\in RZP_0 \\ &\text{and} \\ f(r_j) &= \max\{f(r_i) | r_i \in RZP_0, \; i \in \{0, 1, \cdots, N_{reachable}\}\} \end{aligned} \tag{7}$$

3) INITIAL VISIBLE POPULATION

After forming the reachable population, the next step is to form the visible population. The second population used in the ROA consists of a set of possible solutions visible to the raccoon's eyes, called the Visible Zone Population (*VZP*). The *VZP* is defined by a circle around raccoon with radius *VZR* and consists of $N_{visible}$ possible solution candidates.

$$VZP = \{v_0, v_1, \ldots, v_{N_{visible}}\} \tag{8}$$

where $N_{visible}$ is the number of visible food items and is specified by the user, and $v_i, \forall i \in \{0, 1, \cdots, N_{visible}\}$ are random candidates that represent current food locations. In an N-dimensional optimization problem, v_i is defined as follows:

$$v_i = (y_0, y_1, \ldots, y_{N_{visible}}); \quad \forall i \in \{0, 1, \cdots, N_{visible}\} \tag{9}$$

Additionally, these potential candidates should not be inside the raccoon's reachable zone. In other words, the gap between the members of this population and the location of the raccoon should be greater than the *RZR* and less than or equal to the *VZR*.

Considering $\delta(loc, candidate)$ as the distance of the candidate solution from the current location (loc) of the animal, all the members of the *VZP* should satisfy the following condition:

$$RZR < \delta(loc, candidate) \leq VZR \tag{10}$$

Consequently, the initial visible zone population, denoted by VZP_0, is defined as follows:

$$VZP_0 = \{v_i | i \in \{0, 1, \cdots, N_{visible}\}, \; RZR < \delta(loc_0, v_i) \leq VZR\} \tag{11}$$

As in the previous step, after constructing this population, the best member of this community will be found. This best candidate is called V_{best_0}. Considering the optimization problem as a maximization problem and f as the fitness function to be optimized, we have

$$\begin{aligned} V_{best_0} &= v_j \\ &\text{where} \\ v_j &\in VZP_0 \\ &\text{and} \\ f(v_j) &= \max\{f(v_i) | v_i \in VZP_0, \; i \in \{0, 1, \cdots, N_{visible}\}\} \end{aligned} \tag{12}$$

4) PERSEVERATION PARAMETER

Finally, $n_{perseveration}$ is defined as the perseveration value. This value is used to evaluate the perseveration. A technique based on perseveration is used to avoid local optima and is described in the main loop section in detail.

Initially, in this step, $n_{perseveration}$ is set to zero.

$$n_{perseveration} = 0 \tag{13}$$

Pseudocode 2 details the initialization step.

Pseudocode 2 Step 2. Initialization

1: $loc_0 \leftarrow$ Random Initial Location
2: $G_{opt} \leftarrow loc_0$
3: $RZP_0 \leftarrow$ Initial Reachable Population
4: $R_{best_0} \leftarrow$ Best Reachable Candidate in RZP_0
5: $VZP_0 \leftarrow$ Initial Visible Population
6: $V_{best_0} \leftarrow$ Best Visible Candidate in RZP_0
7: $n_{perseveration} \leftarrow 0$

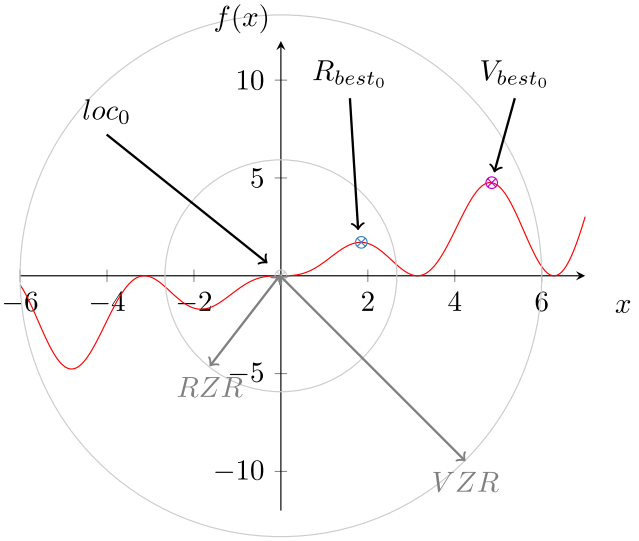

FIGURE 2. A sample initialization of the ROA.

Figure 2 shows one sample initialization of the algorithm. In this example, the initial location of the raccoon is at the origin. Around this position, two populations are constructed. The first population is RZP_0, within the radius RZR. The other population is VZP_0, within the radius VZR. In each iteration, three optimum values exist (loc, R_{best}, and V_{best}); these are shown on the plot as crossed circles.

C. MAIN LOOP

The next portion of the algorithm is the main loop. In this part, the following steps are repeated NI times.

1) RELOCATION TO BEST LOCATION

In each iteration $i, i \in \{0, 1, \cdots, NI\}$, the best value relative to the current location of the raccoon (loc_{i-1}), the best value in the reachable population of the previous iteration ($R_{best_{i-1}}$), and the best value in the visible population of the previous iteration ($V_{best_{i-1}}$) are selected. Then, the raccoon moves to the best position among these three values. Consequently, the raccoon's new location is the location with the best fitness among these three values. Considering the optimization problem as a maximization problem and f as the fitness function to be optimized, this can be notated as follows:

$$\begin{aligned} loc_i &= l_c \\ &\text{where} \\ l_c &\in \{loc_{i-1}, R_{best_{i-1}}, V_{best_{i-1}}\} \\ &\text{and} \\ f(l_c) &= max\{f(h)|h \in \{loc_{i-1}, R_{best_{i-1}}, V_{best_{i-1}}\}. \end{aligned} \tag{14}$$

After relocating the raccoon to the new location, loc_i and G_{opt} are evaluated, and the one with the best fitness is assigned to G_{opt}.

$$\begin{aligned} G_{opt} = (f(G_{opt}) > f(loc_i)) \rightarrow (G_{opt}) \\ \wedge \neg(f(G_{opt}) > f(loc_i)) \rightarrow (loc_i) \end{aligned} \tag{15}$$

2) MIGRATION

After performing relocation, if the location of the animal has not changed for a number of iterations, the raccoon has reached the best food in its local area. However, this best solution could be a local optimum, which could stuck the algorithm. This condition is called *perseveration*.

To evaluate perseveration, a perseveration parameter $n_{perseveration}$ is defined. In the initialization step, $n_{perseveration}$ is set to zero. In each main loop iteration, if the raccoon does not relocate ($loc_i = loc_{i-1}$), the value of this parameter is increased by one ($n_{perseveration} = n_{perseveration} + 1$). On the other hand, if the animal relocates to a new location ($loc_i \neq loc_{i-1}$), this parameter will be reset to zero ($n_{perseveration} = 0$).

$$\begin{aligned} n_{perseveration} = (loc_i = loc_{i-1}) \rightarrow (n_{perseveration} + 1) \\ \wedge (loc_i \neq loc_{i-1}) \rightarrow 0 \end{aligned} \tag{16}$$

To avoid becoming stuck in a local optimum, perseveration is checked in each iteration by comparing the value of $n_{perseveration}$ with the Migration Factor (MF). When $MF = n_{perseveration}$, then migration is performed. Migrating the raccoon means relocating it to a new random location outside its visible zone in the hope of finding a better solution. This random location (loc_i) can be any location inside the problem domain where

$$\delta(loc_i, loc_{i-1}) > VZR \tag{17}$$

After performing migration, $n_{perseveration}$ is reset to zero ($n_{perseveration} = 0$).

Note that the raccoon remembers the best solution that it has ever found as G_{opt}; thus, if migration moves the raccoon to a place with worse solutions, the move will not affect the problem's overall execution, and the raccoon will gradually relocate to better places.

3) GENERATE NEXT GENERATION

At the end of each generation, we build new populations similar to those in the initialization step. The difference here is the location (loc_i) of the raccoon. The new populations are centered around the new location of the animal. However,

Pseudocode 3 Step 3. Main Loop

for $i = 1$ to NI **do**
  $loc_i \leftarrow$best location in loc_{i-1}, $R_{best_{i-1}}$, and $V_{best_{i-1}}$
  if $f(loc_i) > f(G_{opt})$ **then**
    $G_{opt} \leftarrow loc_i$
  end if
  if $loc_i = loc_{i-1}$ **then** {Perseveration}
    $n_{persevere} = n_{persevere} + 1$
  else
    $n_{persevere} = 0$
  end if
  if $n_{persevere} = MF$ **then** {Migration}
    $loc_i \leftarrow$new random location outside of VZP_{i-1}
    $n_{persevere} = 0$
  end if
  $RZP_i \leftarrow$ Reachable Population Around loc_i
  $R_{best_i} \leftarrow$ Best Candidate in RZP_i
  $VZP_i \leftarrow$ Visible Population Around loc_i
  $V_{best_i} \leftarrow$ Best Candidate in VZP
end for

Pseudocode 4 ROA

parameters Definition
Initialization
Main Loop
if $f(loc_{NI}) > f(G_{opt})$ **then**
  Return loc_{NI}
else
  Return G_{opt}
end if

if the animal remains at its previous position (perseveration), we make sure that the members of the new population will have locations that do not repeat those of the previous population:

$$(loc_i = loc_{i-1}) \Leftrightarrow RZP_i \cap RZP_{i-1} = \emptyset$$
$$\wedge$$
$$(loc_i = loc_{i-1}) \Leftrightarrow VZP_i \cap VZP_{i-1} = \emptyset \qquad (18)$$

The pseudocode for this stage is presented in Pseudocode 3.

After repeating the main loop NI times, the best fitness value of G_{opt} and loc_{NI} will be the best solution found during the raccoon's life cycle.

The overall computational process of the ROA is illustrated by a flowchart in Figure 3. Moreover, Pseudocode 4 presents the combination of the different stages to form the complete algorithm.

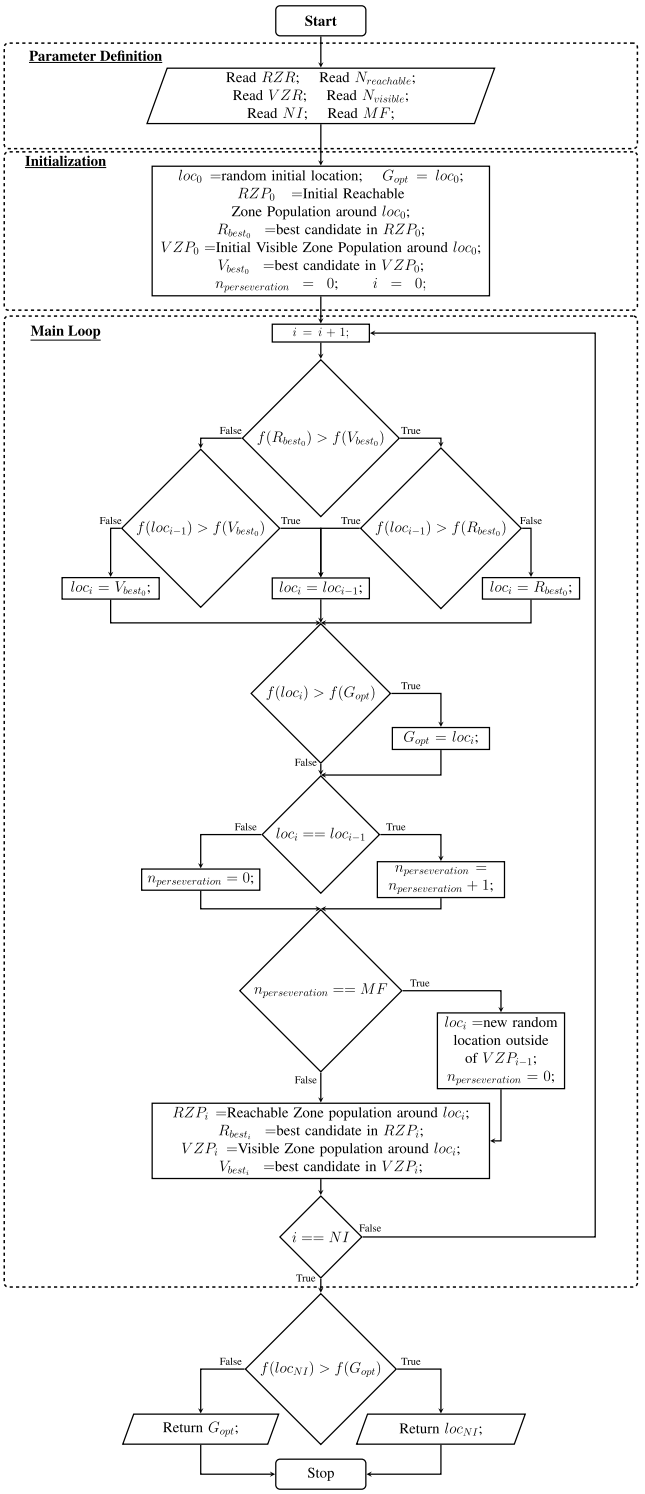

FIGURE 3. ROA.

V. BENCHMARKS AND EXPERIMENTS ON THE ROA

In this section, the efficiency and quality of the proposed ROA are tested using different benchmark functions with different characteristics, and the ROA's results are compared with the results of well-known metaheuristic algorithms including the Genetic Algorithm (GA), Partial Swarm Optimization (PSO), Artificial Bee Colony (ABC), Ant Colony Optimization (ACO), the Cultural Algorithm (CA), the Imperialist Competitive Algorithm (ICA), the Firefly Algorithm (FA), An Improved Teaching-Learning-Based Optimization with Differential (TLBO) and the Invasive Weed Optimization algorithm (IWO).

We chose five different test functions with different characteristics to test the behavior of the proposed algorithm in different situations. First, a simple one-dimensional function is presented to show the structure and the process flow of the algorithm. The other four functions are the Ackley function, Rastrigin function, Grienwank function, and Rotated Hyper-Ellipsoid Function, which are well-known functions that are frequently used to test optimization algorithms. The test functions used as benchmarks in this section stem from the BBOB 2010 [74] and ''Virtual Library of Simulation Experiments, test functions and datasets'' project [75]. The set of optimization test problems in [74] and [75] includes different types of benchmarks with different dimensions, and it has been shown that these functions cover most difficulties in global optimization problems [76]. Hence, these benchmarks can be used to examine optimization algorithms from different aspects. In case of multi dimension functions, the dimension for the experiments are considered as 10.

Each optimization algorithm has its own parameters, which are defined as follows for this study:

- **GA**
  - Selection Method: Tournament
  - Population Size = 100
  - Crossover Percentage = 0.8
  - Mutation Percentage = 0.3
  - Mutation Rate = 0.02
  - Tournament Size = 3
- **PSO**
  - Swarm Size = 100
  - Inertia Weight = 1.0
  - Inertia Weight Damping Ratio = 0.99
  - Personal Learning Coefficient = 1.5
  - Global Learning Coefficient = 2.0
- **ABC**
  - Colony Size = 50
  - Number of Onlooker Bees = 50
  - Abandonment Limit Parameter = 150
  - Acceleration Coefficient Upper Bound = 1
- **ACO**
  - Archive Size = 10
  - Sample Size = 40
  - Intensification Factor = 0.5
  - Deviation-Distance Ratio = 1
- **CA**
  - Population Size = 50
  - Acceptance Ratio = 0.35
  - Alpha = 0.3
  - Beta = 0.5
- **ICA**
  - Population Size = 50
  - Number of Empires/Imperialists = 10
  - Selection Pressure = 1
  - Assimilation Coefficient = 1.5
  - Revolution Probability = 0.05
  - Revolution Rate = 0.1
  - Colony Mean Cost Coefficient = 0.2
- **FA**
  - Number of Fireflies (Swarm Size) = 25
  - Light Absorption Coefficient (γ) = 1
  - Attraction Coefficient Base Value = 2
  - Mutation Coefficient (α) = 0.2
  - Mutation Coefficient Damping Ratio = 0.98
  - Uniform Mutation Range (δ) = $0.05 \times (DomainMax - DomainMin)$
- **TLBO**
  - Population Size = 50
  - Unknown Variables Lower Bound = -10
  - Unknown Variables Upper Bound = 10
- **IWO**
  - Initial Population Size: 10
  - Maximum Population Size = 25
  - Minimum Number of Seeds = 0
  - Maximum Number of Seeds = 5
  - Variance Reduction Exponent = 2
  - Initial Value of Standard Deviation = 0.5
  - Final Value of Standard Deviation = 0.001

In the following, the optimization solutions for the four selected benchmarking functions are obtained using the optimization algorithms described above; then, the results are compared.

A. ONE-DIMENSIONAL FUNCTION

The first benchmark function presented here is a one-dimensional, unimodal function. This function is not comprehensive enough to show the quality and performance of the proposed algorithm. However, its structure allows us to show the actual behavior of the algorithm with different conditions and iterations beyond the complexity of the fitness function. Hence, this function is utilized to illustrate the ROA's structures and techniques in searching for the optimum (minimum) point in the solution domain.

This function is defined as follows:

$$f(x) = (x^2 + x)cos(x) \quad -10 \leq x \leq 10 \tag{19}$$

The known global optimum of this function is $f(x) = -100.22$ for $x = 9.6204$. Figure 4 shows the graphical representation of this function [77].

The ROA is executed to optimize this function. For this execution, the algorithm's parameters are set as shown below.

- $N_{reachable} = 10$
- $RZR = 1$
- $N_{visible} = 5$
- $VZR = 3$
- $NI = 30$
- $MF = 3$

Figure 5 shows the locations of the raccoon in different iterations during a single run. To test the capacity of this algorithm, the initial location (loc_0) was manually set to -10.0,

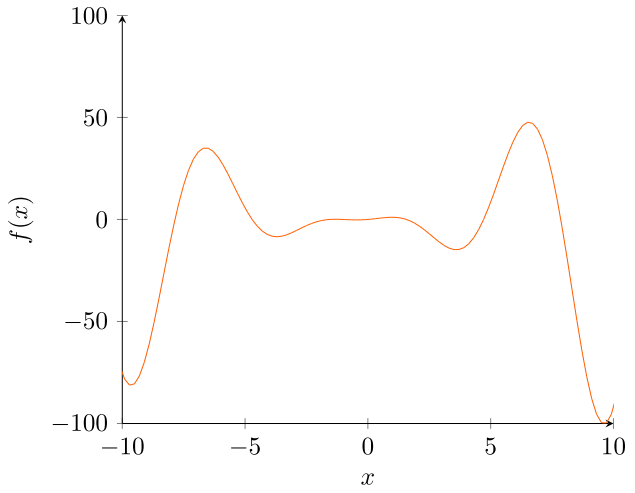

FIGURE 4. $f(x) = (x^2 + x)cos(x)$.

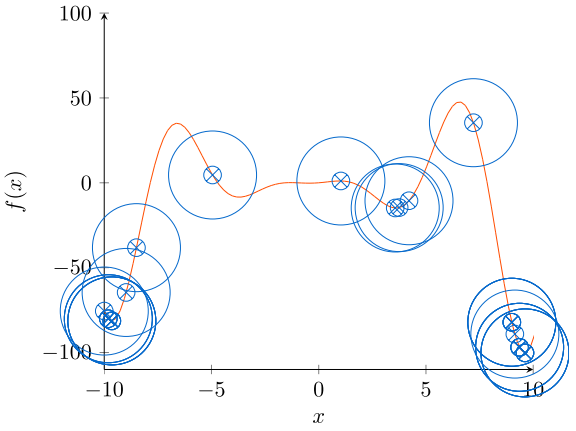

FIGURE 5. Raccoon's locations in different iterations.

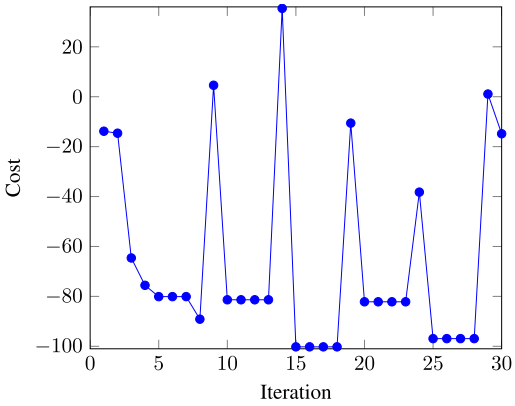

FIGURE 6. Cost plot for locations.

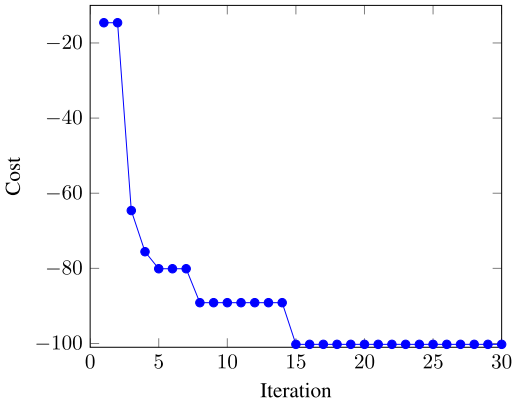

FIGURE 7. Cost plot.

which is far from the known global optimum; moreover, a local optimum exists nearby. Therefore, in this situation, the behavior of the algorithm can be examined when facing a local optimum, which is a bottleneck for optimization problems.

As shown in Figure 5, the raccoon starts at $x = -10.0$. However, after several iterations, it tries to migrate to another random location, which in this sample run is $x = 3.957$. Over the 30 total iterations in this example, the raccoon performs five migrations; at the end, the migrations guide it to the local optimum on the plot, which is $x = 9.620$.

The raccoon stores the best-so-far solution in G_{opt}. However, when the raccoon changes its location to a different location (migrates), it is more likely to find a solution with a worse fitness value in the next round. The locations of the raccoon during different iterations show the impact of migrations on its life cycle. Figure 6 shows the fitness values (costs) of the raccoon's locations in different iterations ($f(loc_i), i \in \{0, 1, \cdots, NI\}$). In this figure, the raccoon's migrations are clearly visible as large jumps. Initially, after each migration, the value of the location is worse, but after a few iterations, it returns to the optimum value. However, in iteration number 8, the raccoon reaches an optimum solution, which is equal to -89.132527, and because no better solution exists within its visible zone, it remains there (perseveration). Nevertheless, it is known that this is only a local optimum for this function. After three iterations ($MF = 3$), the raccoon migrates to another random location. Again, this movement initially causes the solution value to be unfavorable, but after completing several rounds, at iteration 15 it reaches the point whose cost is -100.2206, which is the known global solution for the problem. This scenario shows the impact of migrations, which helps the ROA avoid local optima and helps it to find the global optimum.

As explained before, in each iteration, regardless of where the raccoon is (loc_i), it stores the best solution that it has found so far in G_{opt}. Figure 7 presents the values of G_{opt} in different iterations. In the first iterations, the solutions that it finds are not accurate. However, as the algorithm proceeds, the raccoon moves closer to the global optimum, and at iteration 15, it reaches the best known solution. Because this is the global optimum of the function in this domain, the value of the G_{opt} remains unchanged after this iteration.

This simple example shows the overall behavior of the ROA in solving the optimization problem. As the example shows, the raccoon tries to search different areas of the

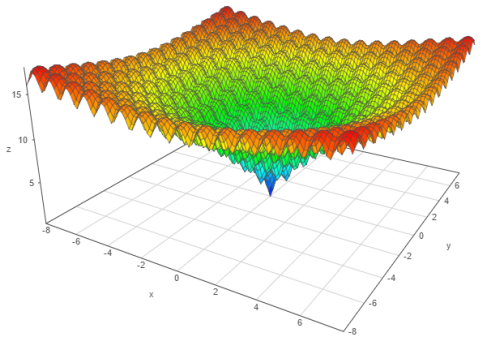

FIGURE 8. Ackley's function.

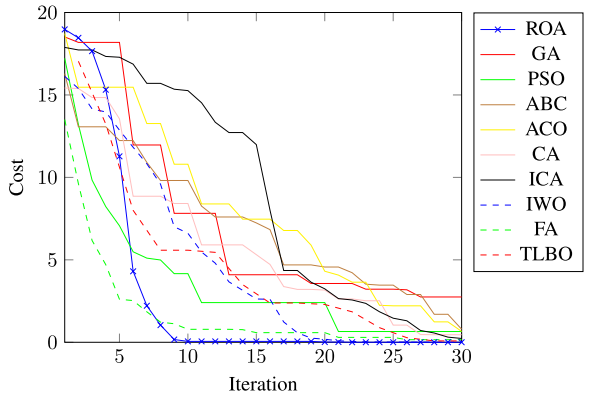

FIGURE 9. Experiments on Ackley's function.

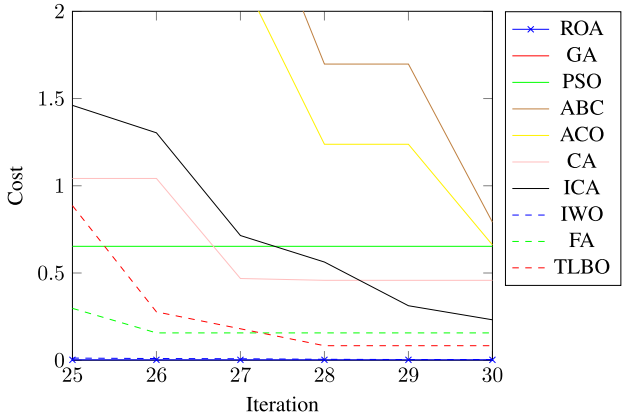

FIGURE 10. Last five iterations in experiments on Ackley's function.

solution domain to find the best food while remembering the previous best location. The following sections show more complicated examples, which consist of well-known benchmark functions for testing optimization algorithms. The ROA and other state-of-the-art metaheuristics are applied to these benchmark functions to compare their abilities from different points of view.

B. ACKLEY'S FUNCTION

One of the most widely used multidimensional test functions for optimization algorithms is Ackley's function [76]. This multimodal function poses a risk for metaheuristics algorithms because it has many local optima where algorithms can easily become stuck. The function is defined as follows:

$$f(x) = -a.e^{-b.\sqrt{\frac{\sum_{i=1}^{N} x_i^2}{n}}} - e^{\frac{\sum_{i=1}^{N} cos(c.x_i)}{n}} + a + e^1$$
$$-32.768 \leq x_i \leq 32.768 \qquad (20)$$

The recommended parameter settings for this function are $a = 20$, $b = 0.2$, and $c = 2\pi$. The known global minimum of this function is $f(x) = 0$ for $x_i = 0, i = 1.\cdots, n$ [75]. Figure 8 shows a visualization of this function for two variables and the recommended parameter set.

To examine the quality of the ROA's solution, the ROA and nine other well-known optimization algorithms (GA, PSO, ABC, ACO, CA, ICA, FA, TLBO and IWO) are implemented and executed under identical conditions to optimize this fitness function. All these algorithms are implemented in MATLAB R2018a x64 and executed on a computer with a 1.90 GHz Intel Core i5-4300U processor and 8 GB of memory.

Each implemented algorithm is executed 30 times, with 30 iterations in each run. The best results obtained from these executions are summarized in Figure 9, which shows the values of the fitness function (cost) in each iteration.

Figure 8 shows that by the last iterations, all the algorithms are close to the best solution. However, because optimization algorithms are approximate in nature, none of these algorithms reach the exact solution in 30 iterations. Therefore, the proximity of the solutions to the real known global optimum in a limited number of iterations is a measurement unit for optimization algorithms. To clarify this, the last five iterations are enlarged in Figure 10.

Figure 10 shows the results obtained from the last five iterations of the different algorithms. According to Figure 9, the ROA starts with results that are worse than those of all the other algorithms. However, this algorithm rapidly moves closer to the solution, and after iteration 25, it approaches the known globally optimum solution. As this plot shows, the TLBO algorithm and the ROA achieve better results than the others. However, the TLBO algorithm's result at iteration 30 is equal to 0.083, while the ROA reaches a point with a cost value of 0.0024. The globally optimum value for Ackley's function is known to be 0. Therefore, on average, the ROA achieves better results than all the other algorithms.

One of the ROA's techniques to avoid local optima is migration. To show how the algorithm acts during different iterations, the fitness values of the raccoon's locations ($f(loc_i)$) are measured in each iteration and plotted in Figure 11, which shows that during this execution, the raccoon performs two migrations. After starting execution, in iteration 12, the raccoon reaches a point with a fitness value of 1.87. Although this point is the minimum fitness value in this area, it is only a local optimum. Therefore, the raccoon stores this point in G_{opt} and migrates to another

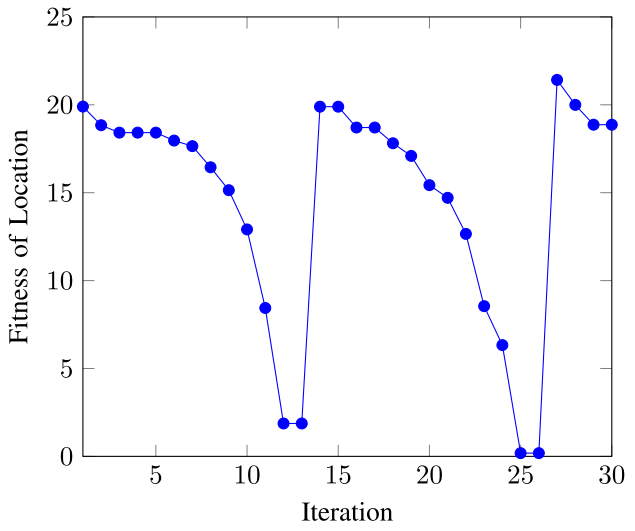

FIGURE 11. Locations of the Raccoon in the Ackley sample run.

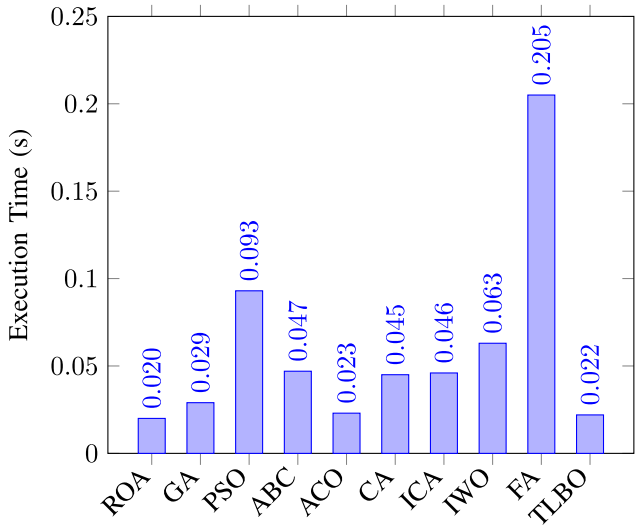

FIGURE 12. Average execution times for Ackley's function.

random location. This random location has a fitness equal to 19.89, which makes it a very bad choice, but after a few iterations, migrations lead the raccoon to a point with fitness equal to 0.0024 at iteration 25, which is the global optimum. However, the animal does not stop here; it performs additional searches in the hope of finding an even better result.

Optimization algorithms are mostly used in large real-world applications; consequently execution times can be considerable. Hence, the execution times of the optimization algorithms can be used as another evaluation measure. This experiment captures the execution times of the different algorithms. The obtained results are presented in Figure 12, which shows that the ROA has the shortest execution time, meaning that it has the fastest performance for solving this optimization problem.

In this section, Ackley's optimization function was optimized using different optimization algorithms; then, the obtained results were compared. Overall, in terms of proximity to the global optimum, the proposed ROA obtained better results in a limited number of iterations than did the other algorithms. The ROA obtained a result of 0.002414, while the TLBO algorithm, which was the next closest, reached 0.08331 and the GA, which obtained the worst result, reached 2.7482.

Moreover, the ROA, which finished in 0.020 seconds, was also the fastest algorithm. Compared to the FA, which finished in 0.205 seconds and was the slowest, the ROA performed 90% faster. Moreover, the TLBO achieved the second-best execution time at 0.022 seconds, which is 9% slower than the ROA.

C. RASTRIGIN FUNCTION

Another benchmark function used to test the Raccoon Optimization Algorithm is the Rastrigin function. This function has been chosen from Real Parameter BBOB 2010 [74] benchmark function set.

The Rastrigin function is a multi-modal functions with adequate global structure for optimization algorithms, and it has many local optima but only one global optimum, which is $f(x_i) = 0$ for $x_i = 0, i = 1, \cdots, d$. Having many local optima and only one global optimum has this function a widely used test function for optimization algorithms [78]. The multidimensional version of this function is defined as follows:

$$f(x) = 10(D - \sum_{i=1}^{D} cos(2\pi z_i)) + ||z||^2 + f_{opt}$$
$$z = \Lambda^{10} T_{asy}^{0.2}(T_{osz}(x - x^{opt})) \quad (21)$$

For more information about the notations and parameter definitions in this function refer to [74]. In this experiment, ten dimensional version of this function ($D = 10$) is considered.

In the problem domain, this function has many small peaks and valleys, making it a serious challenge for optimization algorithms. When optimizing the Rastrigin function, the risk of getting stuck in a local optimum is high, and proximity to a local optimum increases the risk.

Similar to the experiments in the previous sections, the ROA is executed along with the nine other well-known optimization algorithms to optimize this function; then, we show the results to compare the performances of these algorithms. In this experiment, similar to the conditions used to optimize Ackley's function, each algorithm is executed 30 times, with 30 iterations for each run. Then, the best results for each optimizer are calculated. The best results obtained while optimizing this function are shown in Figure 13.

In optimizing Ackley's function, the algorithms' results were similar across all iterations. However, on this function, different algorithms behave differently. Algorithms such as the ROA, and PSO start from a fairly good location and gradually approach the known global optimum. In contrast, other algorithms do not perform well and are not able to reach

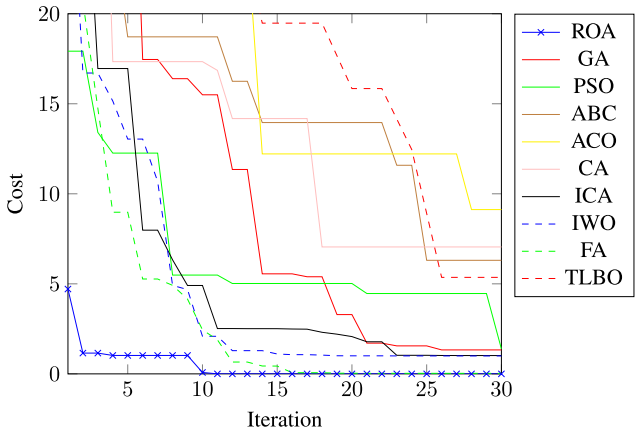

FIGURE 13. Experiments on the Rastrigin function.

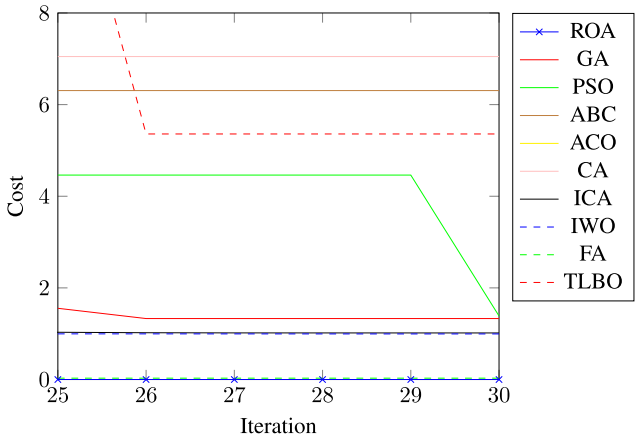

FIGURE 14. Last five iterations in experiments on the Rastrigin function.

the vicinity of the solution. Among all the implemented algorithms, only the ROA, GA, ICA, FA and IWO managed to get close to the known optimum solution in this limited number of iterations. To examine the results in the last iterations, the results are enlarged for the last five iterations as shown in Figure 14.

It is known that the optimum value of the Rastrigin function in this domain is $f(x_i) = 0$ for all $x_i = 0$. The plot in Figure 14 shows that only the ROA and the FA managed to get within a reasonable proximity of the solution. At iteration 30, the fitness values of results obtained for the ROA and FA are 0.00014, and 0.02968, respectively, which means that the ROA improved the result and got much more closer to real known global optimum.

Next, the behavior of the ROA in different iterations is discussed. The raccoon's location during each iteration (loc_i) was captured, and the fitness values of the raccoon's locations at different iterations ($f(loc_i)$) are shown in Figure 15.

As the plot shows, in this run, the raccoon performed four migrations in total. However, it reached the global optimum after only two migrations, in iteration 21. Subsequently, it continued to migrate in the hope of finding better results.

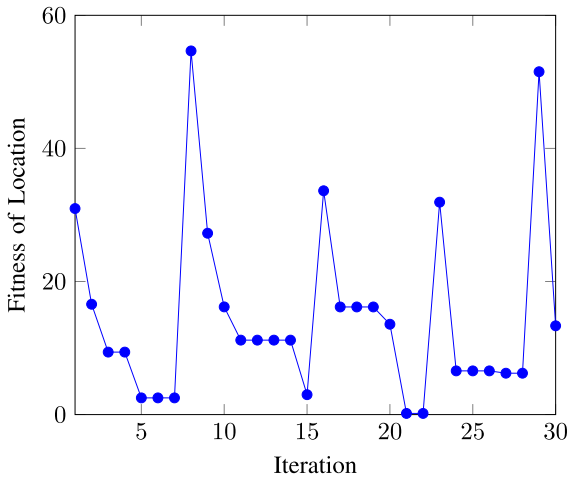

FIGURE 15. Locations of the Raccoon in the Rastrigin sample run.

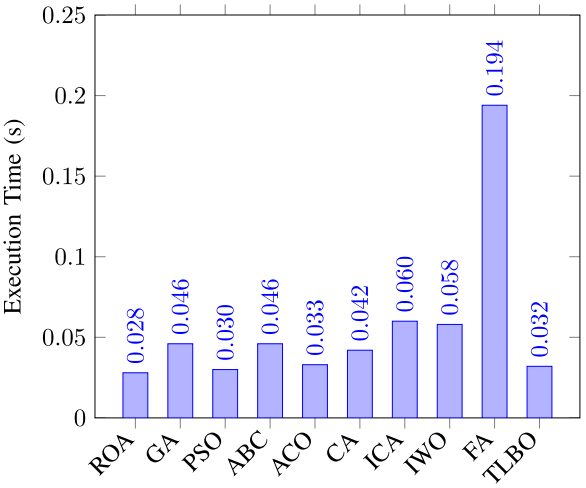

FIGURE 16. Execution times of algorithms on Rastrigin.

Because this function has numerous local optima, after a few iterations, the algorithm became stuck in a local optimum. However, the migrations helped it escape from local optima and continue to try to find the global optimum.

The average execution times of the different algorithms to solve the Rastrigin function are presented in Figure 16, showing that the ROA required the shortest execution time to solve this function.

In the next section, another optimization function the Griewank function is optimized. As with the other benchmarks, the results obtained from these experiments are compared and discussed in detail.

D. GRIEWANK FUNCTION

The other benchmark function used in this paper is the Griewank function. This function was proposed by Griewank [79]. In the optimization field, this function is one of the most challenging and widely used functions for testing

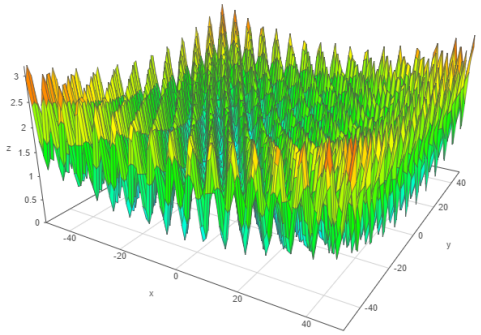

FIGURE 17. Griewank function.

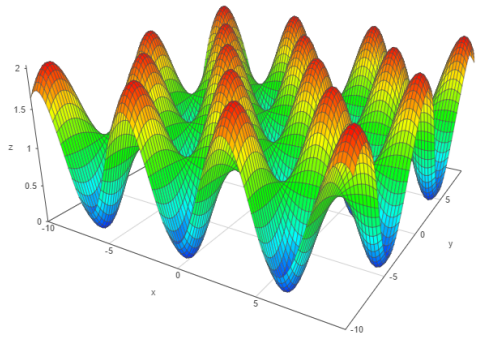

FIGURE 18. Griewank Function in a limited domain.

TABLE 1. Numbers of local minima in $[14, 14]^n$ and $[28, 28]^n$ [81].

n	Number of minima for $[14, 14]^n$	Number of minima for $[28, 28]^n$
1	5	9
2	31	111
3	157	1215

optimization algorithms [80]. The standard Grienwank function is defined below:

$$f(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} cos(\frac{x_i}{\sqrt{i}}) + 1 - 600 \leq x_i \leq 600 \qquad (22)$$

The known global minimum of this function is $f(x) = 0$ for $x_i = 0; i = 1, \cdots, n$. A graphical representation of this function is shown at Figure 17.

One the most important characteristics of this function is the exponential increase in the number of local optima as the function dimensions grow, which makes it challenging for optimization algorithms. To show the local optima situation, the plot Figure 17 is enlarged in Figure 18 to show a limited portion of the domain.

As mentioned above, the minima of this function grow exponentially with the dimension. Table 1 presents the minima for small domains for one, two, and three dimensions [81]. Due to this characteristic, this function is a widely used benchmark for optimization algorithms.

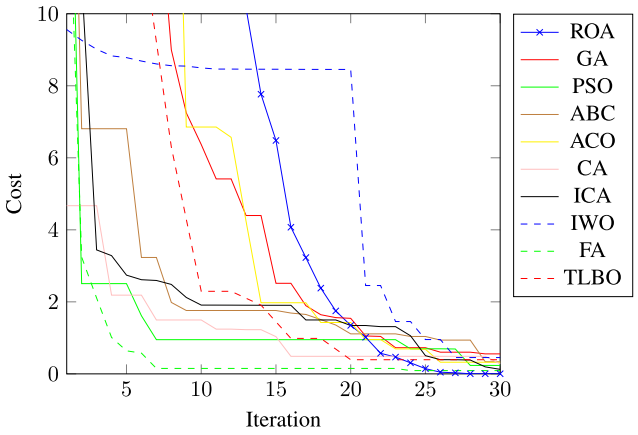

FIGURE 19. Experiments on the Grienwank function.

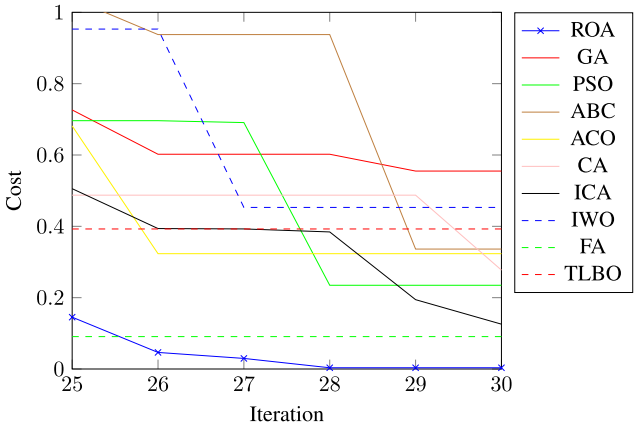

FIGURE 20. Last five iterations in experiments on the Grienwank function.

We optimized the ten dimensional Griewank function using the optimization algorithms introduced earlier. Again, each algorithm is executed 30 times with 30 iterations in each run. The best results obtained in these experiments are captured for each algorithm and shown in Figure 19.

Because the problem domain for this function is wider than that for the other benchmark functions ($[-600, 600]$), all the algorithms were started with relatively large fitness values (> 100). However, they gradually get close to the solution after several iterations. After half of the iterations have been completed, all the algorithms have found solutions close to the global optimum. Therefore, this plot is enlarged to show the differences of the last iterations in detail. Figure 20 shows last five iterations of these experiments.

As Figure 20 shows, similar to the other benchmark function experiments, the ROA obtains the best result among all the compared algorithms. The FA Algorithm achieves second best. After completing the 30 iterations, the ROA reaches a solution with a fitness value equal to 0.003484, whereas the FA's fitness value for the last result is 0.090823, and that of the GA algorithm, which has the solution with the highest fitness value, is 0.554962.

To further evaluate the ROA, the fitness values of the raccoon's locations in each iteration ($f(loc_i)$) are calculated and

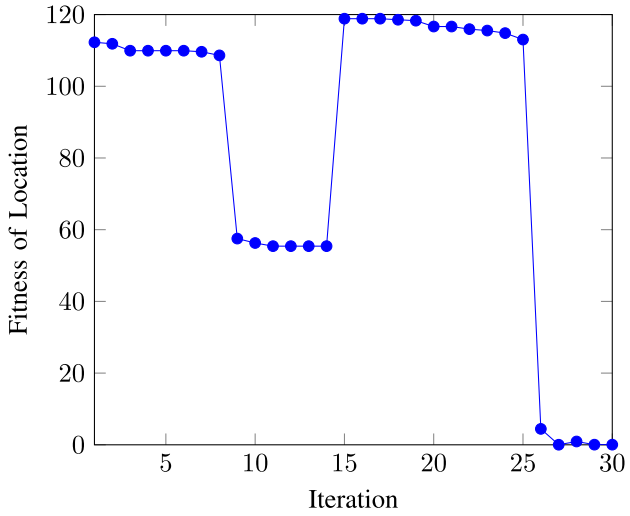

FIGURE 21. Locations of the Raccoon in the Griewank sample run.

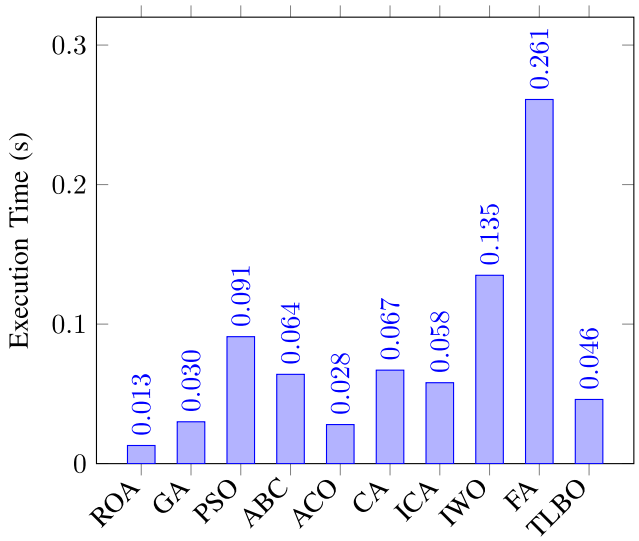

FIGURE 22. Execution times of algorithms on Griewank.

shown in Figure 21. The raccoon performed three migrations. At the beginning of the execution, the raccoon was far from the real solution. However, after 26 iterations, it was close to the desired location, and finally, at iteration 28, it obtained its best solution. In the second half of the raccoon's life, the animal does not perform any migrations because there are many local optima around the global optimum of the function, and when the raccoon reaches this area, it must search many different locations.

Finally, the execution times of the different algorithms are considered, as shown in Figure 22. Similar to all the previous benchmarks, the ROA both requires the least execution time and reaches the best solution in the shortest time.

E. ROTATED HYPER-ELLIPSOID FUNCTION

The Rotated Hyper-Ellipsoid (RHE) is a continues function to test optimization algorithms. This function is an extension

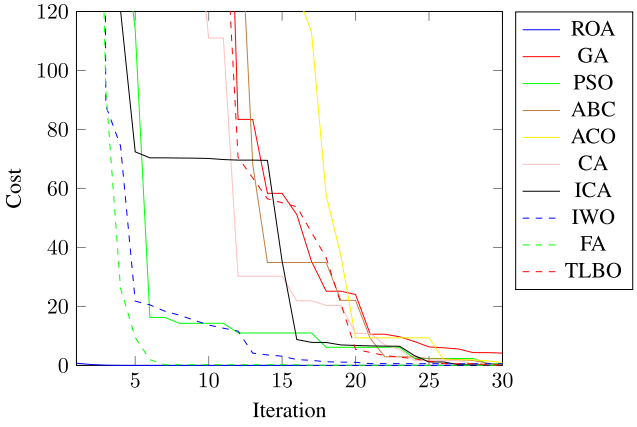

FIGURE 23. Experiments on the rotated hyper-ellipsoid function.

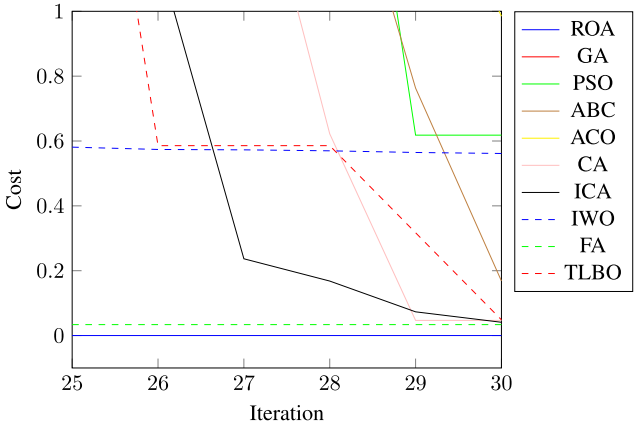

FIGURE 24. Last five iterations in experiments on the Rotated Hyper-Ellipsoid function.

version of two other functions like Axis Parallel Hyper-Ellipsoid and Sum Squares function. RHE mostly is evaluated on the domain $x_i \in [-65.536, 65.536]$ where $i = 1 \ldots d$.

$$f(x) = \Sigma_{i=1}^{n} (\Sigma_{j=1}^{i} x_i)^2$$

Rotated Hyper-Ellipsoid Function is a convex function which is at a risk to stuck in local optimum. The global minimum of the function is at the point $x^* = (0, \ldots, 0)$ with $f(x^*) = 0$.

Rotated Hyper-Ellipsoid Function has been examined in dimension of 10. Unlike the other test functions used in this study, Rotated Hyper-Ellipsoid Function does not have numerous local optimizations. The reason for choosing this kind of function is to show the ability of ROA in finding optimum solutions in both kinds of problems. As the previous functions, the ROA is executed along with aforementioned optimization algorithms.

The result of experiments are depicted in Figure 23. Each algorithm is executed ten times, with 30 iterations for each run, and the best results among these runs are chosen for each algorithm. At the last iterations the result of all algorithms are vary close to each other. Due to this, the 10 last iterations is enlarged in figure 24 to have a better insight into the results.

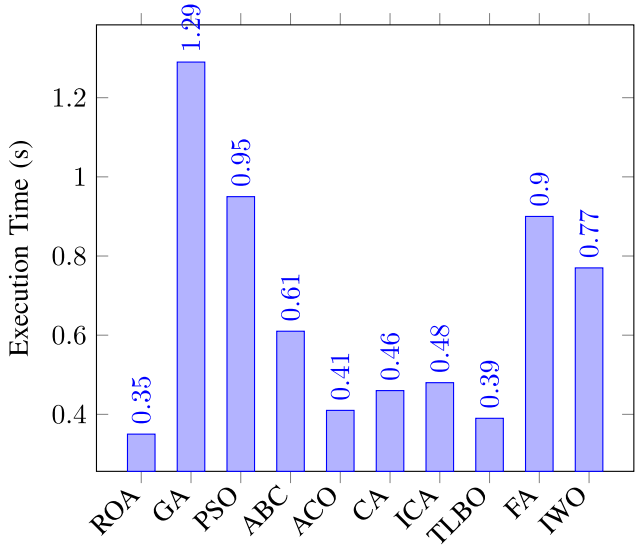

FIGURE 25. Execution times of algorithms on rotated Hyper-Ellipsoid function.

The figure indicates that among all the other algorithms, ROA obtained better results. The FA and ICA algorithms could achieve second best results.

Moreover, ROA was able to obtain a solution with a fitness value equal to 4×10^{-6} in 30 iterations. Whiles the other algorithms like PSO have fitness 0.61 which are the highest value between these algorithms. It shows the ability of ROA in finding optimum results rather than the others in fewer iterations.

Figure 25 shows the ROA is faster than the other aforementioned algorithms in terms of coverage time.

VI. CONCLUSION AND DISCUSSION

In this paper, inspired by the lifestyle of the raccoons, a new optimization algorithm, named ROA (Raccoon Optimization Algorithm) is proposed. Based on the experimental results and problem-solving ability of the meta-heuristic algorithms, we can summarize that ROA is a proper algorithm to solve problems in terms of accuracy and time.

As noted in the paper, ROA is fully shaped based on the promising direction of the meta-heuristic optimization algorithm. ROA enjoys a novel and sufficient neighborhood searches to avoid of becoming stuck in local optimum. Its neighborhood searches are quite different from other well-known algorithms. This algorithm makes use of two different search zones in each iteration. Moreover, it provides a proper technique of searching in the solution state space by searching two different zones, named *visible* and *reachable*, in parallel. By this technique, the number of neighborhoods in each iteration remarkably increase and better results can be achieved. Moreover, due to having visible zones, getting stuck in the local optimum and consequently efforts for getting out of this valley is reduced. These features of ROA have an effective impact on coverage time.

In addition, the raccoon can remember best previously visited location. Then, if it does not manage to find better solutions in the future iterations, it can revert back to the best location. Additionally, this ability helps raccoon to moves purposely toward the optimum solution. All these characteristics set of racoon's life and rummaging style jointly are the basis of ROA which leads to solving problems more simply and accurately.

The ROA eliminates the computational complexity of other previous algorithms such as artificial fish by proposing a simple and easy computing process. All these characteristics are demonstrated by using some well-known benchmark algorithms along with ROA in solving optimization benchmark functions which are challenging enough to examine the abilities of the algorithms. The results of these experiments show that the ROA obtains more accurate solutions in fewer iterations along with a shorter coverage time.

The ROA's execution time could be further reduced by running the algorithm on a parallel architecture. Having n raccoons explore in parallel means while sharing their experiences would be one approach to reducing the execution time. Thus, parallelizing the ROA is the first important future work that we intend to investigate.

Another possibility for our future work is to use machine learning, which might result in the better movement because the racoons could remember and learn from their experiences. Finally, we aim to apply the ROA to multilevel k-way hypergraph partitioning to improve the scheduling and mapping of exa-scale parallel processes into heterogeneous parallel architectures.

REFERENCES

[1] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing* (Natural Computing Series), 2nd ed. Berlin, Germany: Springer, 2015.

[2] B. Bontoux and D. Feillet, "Ant colony optimization for the traveling purchaser problem," *Comput. Oper. Res.*, vol. 35, no. 2, pp. 628–637, Feb. 2008.

[3] K. V. Singh and Z. Raza, "A ga based job scheduling strategy for computational grid," in *Proc. Int. Conf. Adv. Comput. Eng. Appl. (ICACEA)*, Mar. 2015, pp. 29–34.

[4] Q. Lin, Q. Zhu, P. Huang, J. Chen, Z. Ming, and J. Yu, "A novel hybrid multi-objective immune algorithm with adaptive differential evolution," *Comput. Oper. Res.*, vol. 62, pp. 95–111, Oct. 2015.

[5] E.-G. Talbi, *Metaheuristics: From Design to Implementation*, vol. 74. Hoboken, NJ, USA: Wiley, 2009.

[6] S. H. A. Kaboli, J. Selvaraj, and N. A. Rahim, "Long-term electric energy consumption forecasting via artificial cooperative search algorithm," *Energy*, vol. 115, pp. 857–871, Nov. 2016.

[7] A. Rafieerad *et al.*, "Toward improved mechanical, tribological, corrosion and *in-vitro* bioactivity properties of mixed oxide nanotubes on Ti–6Al–7Nb implant using multi-objective PSO," *J. Mech. Behav. Biomed. Mater.*, vol. 69, pp. 1–18, May 2017.

[8] S. S. Sebtahmadi, H. B. Azad, S. H. A. Kaboli, M. D. Islam, and S. Mekhilef, "A PSO-DQ current control scheme for performance enhancement of Z-source matrix converter to drive IM fed by abnormal voltage," *IEEE Trans. Power Electron.*, vol. 33, no. 2, pp. 1666–1681, Feb. 2018.

[9] S. H. A. Kaboli, A. Fallahpour, J. Selvaraj, and N. Rahim, "Long-term electrical energy consumption formulating and forecasting via optimized gene expression programming," *Energy*, vol. 126, pp. 144–164, May 2017.

[10] A. Rafieerad *et al.*, "GEP-based method to formulate adhesion strength and hardness of nb PVD coated on Ti–6Al–7Nb aimed at developing mixed oxide nanotubular arrays," *J. Mech. Behav. Biomed. Mater.*, vol. 61, pp. 182–196, 2016.
[11] X.-S. Yang, "Harmony search as a metaheuristic algorithm," in *Musicinspired Harmony Search Algorithm*. Berlin, Germany: Springer, 2009, pp. 1–14.
[12] N. Mladenović and P. Hansen, "Variable neighborhood search," *Comput. Oper. Res.*, vol. 24, no. 11, pp. 1097–1100, 1997.
[13] D. B. Fogel, *Evolutionary Computation: The Fossil Record*. Hoboken, NJ, USA: Wiley, 1998.
[14] A. Wedyan, J. Whalley, and A. Narayanan, "Hydrological cycle algorithm for continuous optimization problems," *J. Optim.*, vol. 2017, Nov. 2017, Art. no. 3828420.
[15] K. Tamura and K. Yasuda, "The spiral optimization algorithm: Convergence conditions and settings," *IEEE Trans. Syst., Man, Cybern. Syst.*, to be published.
[16] A. R. Pouya, M. Solimanpur, and M. J. Rezaee, "Solving multi-objective portfolio optimization problem using invasive weed optimization," *Swarm Evol. Comput.*, vol. 28, pp. 42–57, Jun. 2016.
[17] T. R. Biyanto *et al.*, "Killer whale algorithm: An algorithm inspired by the life of killer whale," *Procedia Comput. Sci.*, vol. 124, pp. 151–157, Dec. 2017.
[18] H. F. Farahani and F. Rashidi, "An improved teaching-learning-based optimization with differential evolution algorithm for optimal power flow considering HVDC system," *J. Renew. Sustain. Energy*, vol. 9, no. 3, p. 035505, 2017.
[19] E. Emary *et al.*, "Multi-objective gray-wolf optimization for attribute reduction," *Procedia Comput. Sci.*, vol. 65, pp. 623–632, Jan. 2015.
[20] B. Xing and W.-J. Gao, "Fruit fly optimization algorithm," in *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms*. Cham, Switzerland: Springer, 2014, pp. 167–170.
[21] K. Sörensen, "Metaheuristics—The metaphor exposed," *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 3–18, 2015.
[22] C.-R. Wang, C.-L. Zhou, and J.-W. Ma, "An improved artificial fish-swarm algorithm and its application in feed-forward neural networks," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 5, Aug. 2005, pp. 2890–2894.
[23] C. Darwin, *The Origin of Species*. New York, NY, USA: Modern Library, 1859.
[24] A. M. Turing, "Intelligent machinery, a heretical theory," *Philosophia Math.*, vol. 4, no. 3, pp. 256–260, 1996, doi: 10.1093/philmat/4.3.256.
[25] H. J. Bremermann, "Optimization through evolution and recombination," *Self-Organizing Syst.*, vol. 93, p. 106, May 1962.
[26] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. Hoboken, NJ, USA: Wiley, 1966.
[27] J. H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM J. Comput.*, vol. 2, no. 2, pp. 88–105, 1973.
[28] I. Rechenberg, *Evolution Strategy: Optimization of Technical Systems by Means of Biological Evolution*, vol. 104. Stuttgart, Germany: Fromman-Holzboog, 1973.
[29] H.-P. Schwcfcl *et al.*, *Evolution and Optimum Seeking*, vol. 1515. New York, NY, USA: Wiley, 1995.
[30] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, vol. 11. New York, NY, USA: Oxford Univ. Press, 1996.
[31] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary Computation 1: Basic Algorithms and Operators*, vol. 1. Boca Raton, FL, USA: CRC Press, 2000.
[32] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary Computation 2: Advanced Algorithms and Operators*, vol. 1. Boca Raton, FL, USA: CRC Press, 2000.
[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simmulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[34] B. Suman and P. Kumar, "A survey of simulated annealing as a tool for single and multiobjective optimization," *J. Oper. Res. Soc.*, vol. 57, no. 10, pp. 1143–1160, Oct. 2006.
[35] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, 1986.
[36] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. dissertation, Politecnico di Milano, Milan, Italy, 1992.
[37] V. Selvi and D. R. Umarani, "Comparative analysis of ant colony and particle swarm optimization techniques," *Int. J. Comput. Appl. (0975–8887)*, vol. 5, no. 4, 2010.
[38] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Int. Symp. Micro Mach. Hum. Sci.*, New York, NY, USA, vol. 1, Oct. 1995, pp. 39–43.
[39] M. Schmitt and R. Wanka, "Particle swarm optimization almost surely finds local optima," *Theor. Comput. Sci.*, vol. 561, pp. 57–72, Jan. 2015.
[40] M. R. Bonyadi and Z. Michalewicz, "Particle swarm optimization for single objective continuous space problems: A review," *Evol. Comput.*, vol. 25, no. 1, pp. 1–54, 2017.
[41] R. Storn and K. Price, "Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces," ICSI Berkeley, Berkeley, CA, USA, Tech. Rep., 1995, vol. 3.
[42] D. H. Wolper and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[43] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *J. Simul.*, vol. 76, no. 2, pp. 60–68, Feb. 2001.
[44] D. Manjarres *et al.*, "A survey on applications of the harmony search algorithm," *Eng. Appl. Artif. Intell.*, vol. 26, no. 8, pp. 1818–1831, 2013.
[45] S. Nakrani and C. Tovey, "On honey bees and dynamic server allocation in Internet hosting centers," *Adapt. Behav.*, vol. 12, nos. 3–4, pp. 223–240, 2004.
[46] D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi, "Bees algorithm," Manuf. Eng. Centre, Cardiff Univ., Cardiff, U.K., Tech. Note, 2005.
[47] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Dept. Comput. Eng., Erciyes Univ., Kayseri, Turkey, Tech. Rep.-06, 2005.
[48] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: Artificial bee colony (ABC) algorithm and applications," *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, 2014.
[49] M. S. Kiran, H. Hakli, M. Gunduz, and H. Uguz, "Artificial bee colony algorithm with variable search strategy for continuous optimization," *Inf. Sci.*, vol. 300, pp. 140–157, Apr. 2015.
[50] M.-H. Horng, "Multilevel thresholding selection based on the artificial bee colony algorithm for image segmentation," *Expert Syst. Appl.*, vol. 38, no. 11, pp. 13785–13791, 2011.
[51] A. Banharnsakun, T. Achalakul, and B. Sirinaovakul, "The best-so-far selection in artificial bee colony algorithm," *Appl. Soft Comput.*, vol. 11, pp. 2888–2901, Mar. 2011.
[52] X.-S. Yang, "Firefly algorithm," in *Nature-Inspired Metaheuristic Algorithms*, vol. 20. Hoboken, NJ, USA: Wiley, 2008, pp. 79–90.
[53] N. Ali, M. A. Othman, M. N. Husain, and M. H. Misran, "A review of firefly algorithm," *ARPN J. Eng. Appl. Sci.*, vol. 9, no. 10, pp. 1732–1736, 2014.
[54] L. Zhang, L. Liu, X.-S. Yang, and Y. Dai, "A novel hybrid firefly algorithm for global optimization," *PLoS ONE*, vol. 11, no. 9, p. e0163230, 2016.
[55] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in *Proc. World Congr. Nature, Biologic. Inspired Comput. (NaBIC)*, 2009, pp. 210–214.
[56] S. Walton, O. Hassan, K. Morgan, and M. Brown, "Modified cuckoo search: A new gradient free optimisation algorithm," *Chaos, Solitons Fractals*, vol. 44, no. 9, pp. 710–718, 2011.
[57] X.-S. Yang and S. Deb, "Multiobjective cuckoo search for design optimization," *Comput. Oper. Res.*, vol. 40, no. 6, pp. 1616–1624, Jun. 2013.
[58] L. Huang, S. Ding, S. Yu, J. Wang, and K. Lu, "Chaos-enhanced cuckoo search optimization algorithms for global optimization," *Appl. Math. Model.*, vol. 40, nos. 5–6, pp. 3860–3875, 2016.
[59] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimizatio (NICSO)*. Berlin, Germany: Springer, 2010, pp. 65–74.
[60] X.-S. Yang and X. He, "Bat algorithm: Literature review and applications," *Int. J. Bio-Inspired Comput.*, vol. 5, no. 3, pp. 141–149, 2013.
[61] M. Dixit, N. Upadhyay, and S. Silakari, "An exhaustive survey on nature inspired optimization algorithms," *Int. J. Softw. Eng. Appl.*, vol. 9, no. 4, pp. 91–104, 2015.
[62] S. Binitha and S. S. Sathya, "A survey of bio inspired optimization algorithms," *Int. J. Soft Comput. Eng.*, vol. 2, no. 2, pp. 137–151, 2012.
[63] R. A. Mahale and S. Chavan, "A survey: Evolutionary and swarm based bio-inspired optimization algorithms," *Int. J. Sci. Res. Publications*, vol. 2, no. 12, pp. 1–6, 2012.
[64] M. H. Yar, V. Rahmati, and H. R. D. Oskouei, "A survey on evolutionary computation: Methods and their applications in engineering," *Mod. Appl. Sci.* vol. 10, no. 11, p. 131, 2016.

[65] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Trans. Evol. Comput.*, vol. 20, no. 4, pp. 606–626, Aug. 2016.
[66] S. Li, L. Kang, and X.-M. Zhao, "A survey on evolutionary algorithm based hybrid intelligence in bioinformatics," *BioMed Res. Int.*, vol. 2014, Jan. 2014, Art. no. 362738.
[67] R. Devi, E. Barlaskar, O. Devi, S. Medhi, and R. Shimray, "Survey on evolutionary computation tech techniques and its application in different fields," *Int. J. Inf. Theory*, vol. 3, no. 3, pp. 73–82, 2014.
[68] A. M. Shaheen, S. R. Spea, S. M. Farrag, and M. A. Abido, "A review of meta-heuristic algorithms for reactive power planning problem," *AIN Shams Eng. J.*, vol. 9, no. 2, pp. 215–231, 2015.
[69] G. A. Schweickardt, V. Miranda, and G. Wiman, "A comparison of meta-heuristics algorithms for combinatorial optimization problems. Application to phase balancing in electric distribution systems," *Latin Amer. Appl. Res.*, vol. 41, no. 2, pp. 113–120, 2011.
[70] G. A. El-N. A. Said, A. M. Mahmoud, and El-S. M. El-Horbaty. (2014). "A comparative study of meta-heuristic algorithms for solving quadratic assignment problem." [Online]. Available: https://arxiv.org/abs/1407.4863
[71] S. H. A. Kaboli, J. Selvaraj, and N. Rahim, "Rain-fall optimization algorithm: A population based algorithm for solving constrained optimization problems," *J. Comput. Sci.*, vol. 19, pp. 31–42, Mar. 2017.
[72] M. Elbroch and K. Rinehart, *Behavior of North American Mammals*. Boston, MA, USA: Houghton Mifflin Harcourt, 2011.
[73] S. Dehaene, *The Number Sense: How the Mind Creates Mathematics*. New York, NY, USA: Oxford Univ. Press, 2011.
[74] S. Finck *et al.*, "Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions," Res. Center PPE, Tech. Rep. 2009/20, Feb. 2010.
[75] S. Surjanovic and D. Bingham, "Virtual library of simulation experiments: Test functions and datasets," Simon Fraser Univ., Burnaby, BC, Canada, 2013. Accessed: May 13, 2013. [Online]. Available: http://www.sfu.ca/~ssurjano/optimization.html
[76] M. Molga and C. L. Smutnicki, "Test functions for optimization needs," in *Test Functions for Optimization Needs*, vol. 101. 2005.
[77] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*. Hoboken, NJ, USA: Wiley, 2004.
[78] A. Torn and A. Zilinskas, *Global Optimization*. New York, NY, USA: Springer-Verlag, s1989.
[79] A. O. Griewank, "Generalized descent for global optimization," *J. Optim. Theory Appl.*, vol. 34, no. 1, pp. 11–39, 1981.
[80] M. Locatelli, "A note on the Griewank test function," *J. Global Optim.*, vol. 25, no. 2, pp. 169–174, 2003.
[81] H. Cho, F. Olivera, and S. D. Guikema, "A derivation of the number of minima of the Griewank function," *Appl. Math. Comput.*, vol. 204, no. 2, pp. 694–701, 2008.

SINA ZANGBARI KOOHI received the B.Sc. degree in computer science from Mazandaran University and the M.Sc. degree in computer science from Tabriz University. He is currently pursuing the Ph.D. degree in parallel and distributed computing with UPM. His research interests include distributed systems, parallel computing, graph theory, and artificial intelligence. He also has experience in robotics and acquired several rewards in national robotic tournaments.

NOR ASILAH WATI ABDUL HAMID received the Ph.D. degree from The University of Adelaide in 2008. She has been a Visiting Scholar with the High Performance Computing Lab, The George Washington University, USA, for two years. She is currently a Senior Lecturer with the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia, where she is also an Associate Researcher and a Coordinator of high speed machine with the Institute for Mathematical Research (INSPEM). She also holds an associate position with the Distributed and High Performance Computing Group, focusing on high-performance distributed and parallel computing technologies and applications. Her research interests are in parallel and distributed computing, cluster computing, distributed information systems, and other applications of high-performance computing.

MOHAMED OTHMAN received the Ph.D. degree (Hons.) from Universiti Kebangsaan Malaysia. He is currently a Professor with the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, where he is also an Associate Researcher with the Computational Science and Mathematical Physics Laboratory, Institute of Mathematical Research. He has authored over 160 international journals and 230 proceeding papers. His main research interests are in the fields of high-speed network, parallel and distributed algorithms, software-defined networking, network design and management, wireless network including MPDU- and MSDU-frame aggregation, TCP performance, MAC layer, resource management, traffic monitoring, and scientific telegraph equation and modeling. He received the Best Ph.D. Thesis Award in 2000 from the Sime Darby Malaysia and the Malaysian Mathematical Science Society.

GAFURJAN IBRAGIMOV received the Ph.D. and D.Sc. degrees from the National University of Uzbekistan in 1991 and 2006, respectively. He gave a series of talks on differential games as an Invited Professor with the University of Auckland in 2015. He is currently an Associate Professor with the Department of Mathematics, Faculty of Science, Universiti Putra Malaysia, Malaysia. He is also an Associate Researcher with INSPEM, Universiti Putra Malaysia, and also with the Institute of Mathematics, National University of Uzbekistan. His research interests are differential games, control theory, and simultaneous games. He is a Guest Editor of the Special Issue on Differential Games and Discrete Dynamics in Applied Sciences of the *Discrete Dynamics in Nature and Society Journal* and an Editorial Board Member of the *Nonlinear Analysis and Differential Equations Journal*.