IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received June 8, 2021, accepted June 26, 2021, date of publication July 14, 2021, date of current version August 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3097182

Particle Swarm Optimization Algorithm for Detecting Distributed Predicates

ESLAM AL MAGHAYREH[1,2], (Member, IEEE), HABIB DHAHIRI[1], FAHAD ALBOGAMY[3], MOHAMAD MAHMOUD AL RAHHAL[1], (Member, IEEE), AWAIS MAHMOOD[1], ESAM OTHMAN[1], AND WAIL S. ELKILANI[1]

[1]Faculty of Applied Computer Science, King Saud University, Riyadh 11451, Saudi Arabia
[2]Department of Computer Science, Yarmouk University, Irbid 21163, Jordan
[3]Computer Sciences Program, Turabah University College, Taif University, Ta'if 26571, Saudi Arabia

Corresponding author: Eslam Al Maghayreh (ealmaghayreh@ksu.edu.sa)

This work was supported by the Deanship of Scientific Research at King Saud University through the research group under Grant RG-1438-071.

ABSTRACT Metaheuristic algorithms are widely used to solve NP-complete problems in several domains. Distributed predicates detection is a fundamental distributed systems problem that has many useful applications. The problem of distributed predicates detection, in general, is known to be an NP-complete problem. In this paper, we developed a detection algorithm inspired by the particle swarm optimization algorithm, one of the well-known metaheuristic algorithms applied to solve problems in several domains. The proposed detection algorithm deal with distributed predicates under the possibly modality. We compared the performance of the proposed distributed predicates detection algorithm with several other detection algorithms. The experimental results reveal the effectiveness of the suggested distributed predicates detection algorithm.

INDEX TERMS Artificial intelligence, computational intelligence, debugging, distributed predicates detection, distributed systems, monitoring, particle swarm optimization, runtime verification, testing.

I. INTRODUCTION

Distributed predicates detection (DPD) is a fundamental distributed systems problem. Several researchers have developed distributed predicates detection algorithms that can be exploited for several purposes [1]–[6], [6]–[12]. For example, to implement the following basic command in debugging, "stop the program when condition C is true," in a distributed environment, you can represent condition C as a distributed predicate, and then you can use a distributed predicates detection algorithm to detect its satisfaction [13].

Sensors can provide rich context information for pervasive applications that are typically designed to be context-aware. Context-awareness allows applications to adapt to the dynamic pervasive computing environments in an intelligent manner [14], [15]. Detecting distributed predicates over asynchronous computations can be exploited to achieve context-awareness in pervasive environments [6], [11], [14], [16].

The associate editor coordinating the review of this manuscript and approving it for publication was Ziyan Wu.

Owing to concurrency in distributed systems, in which many processes are running simultaneously, it is generally challenging to detect a distributed predicate. Concurrency results in an exponentially large search space for the DPD problem. In fact, this problem has been already proved to be NP-complete [13]. The algorithms existing in the literature for DPD work efficiently for some types of distributed predicates [13], [17].

Several meta-heuristic based algorithms have been proposed to tackle NP-complete problems. Particle swarm optimization (PSO) is a well-known meta-heuristic algorithm that has been exploited widely to solve optimization problems [18]–[24]. In this paper, we will develop a DPD algorithm inspired by the PSO algorithm. In [25], the authors claim that PSO has the same effectiveness (finding the true global optimal solution) as Genetic algorithms but with significantly better computational efficiency (less function evaluations). Compared with other metaheuristic algorithms like Genetic algorithms, PSO works on two populations. This allows greater diversity and exploration over a single population. For these reasons, we have considered the use of the PSO algorithm in this research paper.

The remaining part of this paper involves the following sections. Section II formally defines the principal terms to be used during the presentation of this paper. Section III presents some of the main contributions in the domain of DPD. An overview of the PSO algorithm is given in Section IV. After that, the proposed DPD algorithm exploiting the PSO algorithm will be detailed in Section V. Section VI presents the results of the experiments accomplished to evaluate the proposed DPD algorithm. Finally, conclusions and future works are presented in Section VII.

II. A FORMAL MODEL

We have dedicated this section to describe the formal model used to precisely represent a distributed program run. We will also formally define the terms we are going to use in describing the proposed distributed predicates detection (DPD) algorithm.

An asynchronous message-passing distributed program involves n concurrent processes $(P_0, P_1, \ldots, P_{n-1})$ connected through a set of channels. These processes do not share a memory or a clock.

Each process P_i executes a sequence of events $(e_{i0}, e_{i1}, \ldots)$ and generates a sequence of local states $(s_{i0}, s_{i1}, \ldots)$. An event is a result of executing some statement in the corresponding distributed program. An event moves a process from one local state to another. A given local state s_{ij} involves the values of process P_i variables, including program counter (PC), immediately after executing the event that moves process P_i to local state s_{ij}.

Figure 1 is a graphical depiction of a simple distributed program's execution consisting of three processes (P_0, P_1, and P_2). For example, process P_1 has executed events $(e_{10}, e_{11}, e_{12}, e_{13})$ and generates the following sequence of local states $(s_{10}, s_{11}, s_{12}, s_{13})$. In Figure 1, small filled circles are used to depict events and circles are used to depict local states. s_{10} is the initial local state of P_1. The execution of event e_{11} moves process P_1 from local state s_{10} to local state s_{11}. In the context of our problem, we need only to keep a record of the variables used in expressing the predicates to be detected. Consequently, we are not showing all of the details involved in a local state in general.

A message is depicted as a directed edge connecting the send and receive events. In Figure 1, the directed edge between e_{01} and e_{11} means that process P_0 has sent a message to process P_1. An event e_{ij} is said to have happened-before event e_{kl} $(e_{ij} \rightarrow e_{kl})$ if the corresponding run contains a directed path going from e_{ij} to e_{kl} [26]. In Figure 1, $e_{00} \rightarrow e_{11}$. e_{10} does not happen before e_{20}, and e_{20} does not happen before e_{10}. e_{10} and e_{20} are called concurrent events.

A ***cut*** (subset) of a given run is called ***consistent*** if and only if the following requirement is fulfilled:

For any two events e_{ij} and e_{kl}, if e_{kl} is in the cut and $e_{ij} \rightarrow e_{kl}$, then e_{ij} is also in the cut.

For example, the dashed line drawn in Figure 1 is used to display a consistent cut involving the set of events $\{e_{00}, e_{01}, e_{02}, e_{10}, e_{11}, e_{20}\}$.

A global state comprises one local state from each process. Namely, it involves the local states reached by executing the events of a given consistent cut. The consistent cut rendered using a dashed line in Figure 1 is mapped to a global state G_1 of the run. The global state G_1 involves the following local states $\{s_{02}, s_{11}, s_{20}\}$. $G_0 = \{s_{00}, s_{10}, s_{20}\}$ is the initial global state, and $G_f = \{s_{02}, s_{13}, s_{22}\}$ is the final global state of the run in Figure 1. The global states of a run produce a structure recognized as the state lattice [27].

A distributed predicate is a Boolean expression expressed using the variables of two or more processes. To check whether a distributed predicate has been satisfied at some point in a distributed programs run, we need to keep evaluating the predicate until we find a global state satisfying the predicate. Figure 2 portrays the DPD environment [17].

The number of possible global states in a given run is usually huge. For a run with n processes, where each process has gone through m local states, we will have $O(m^n)$ global states. Consequently, DPD is a difficult time-consuming task in general. In [13], the authors proved that DPD is an NP-complete problem in general.

Some researchers have succeeded in developing efficient detection algorithms for certain types of distributed predicates. More details about these algorithms will be given in the following section.

III. RELATED WORK

There is no efficient general algorithm to detect any distributed predicate. Researchers have developed several algorithms for detecting certain types of distributed predicates. A well-known type of distributed predicates that are easy to detect is called stable predicates. When a stable predicate becomes true, it will remain true. This property of stable predicates facilitates the development of detection algorithms for such a kind of predicates. The global snapshot algorithm presented in [28], [29] can be easily used to detect stable predicates. Termination and deadlock detection are two well-known scenarios where we can use stable predicates.

Other researchers exploit the structure of some distributed predicates in developing efficient algorithms for detecting certain types of distributed predicates. For example, in [30] the authors developed an efficient algorithm for detecting conjunctive predicates. The time complexity of their algorithm is $O(n^2m)$ (assuming we have n processes and m local states in any process).

In [3], [31]–[33], the authors have applied genetic algorithms, harmony search, artificial bee colony, and simulated annealing to develop DPD algorithms that can be used for DPD under the possibly modality. The experimental results demonstrate that these algorithms are more powerful than classical algorithms based on state enumeration.

In [34]–[36], the concept of computation slicing has been utilized to reduce the size of the search space to be considered by the DPD algorithms. A slice of a distributed run concerning a specific predicate is the subset of the global states of the run that contains the global states in which the desired

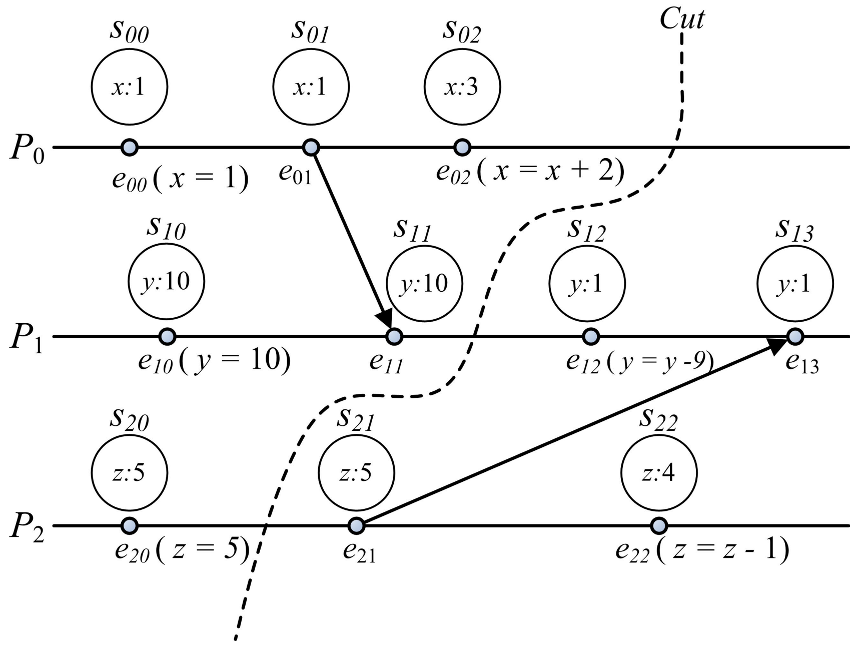

FIGURE 1. A run of a distributed program.

predicates might be true. Consequently, only the global states of the slice have to be examined throughout the detection process. As a result, the detection time will be significantly reduced [34]–[36].

In [37]–[39], the authors have succeeded in exploiting the concept of atomicity in reducing the size of the search space to be examined during the process of detecting distributed predicates. According to their work, there is no need to consider the local state generated after the execution of every single event. At an abstract level, a distributed program executes a number of high-level atomic actions. Each atomic action involves a set of events. In general, we need to consider local states generated after executing atomic actions. This results in a significant reduction in the size of the search space to be processed by the DPD algorithms.

In this paper, we exploited the well-known particle swarm optimization (PSO) algorithm in designing an efficient DPD algorithm. An overview of the basic PSO algorithm and more details about the proposed PSO-based DPD algorithm will be addressed in subsequent sections.

IV. OVERVIEW OF THE PARTICLE SWARM OPTIMIZATION (PSO) ALGORITHM

The particle swarm optimization (PSO) algorithm is a population-based meta-heuristic algorithm developed by Kennedy and Eberhart [40]. The PSO algorithm is inspired by the behavior of the groups of some animals like birds flocks or fish schools. Each member of the population is referred to as a particle. Each particle is characterized by its velocity and position. Algorithm 1 abstractly describes the basic PSO algorithm which consists of the following steps:

Step 1: The velocity v_i, and the position x_i of each particle p_i are randomly initialized. The best local position p_i^l reached by particle p_i is initialized to x_i (See step 1 of Algorithm 1). The position of each particle encodes a possible solution to the problem under consideration.

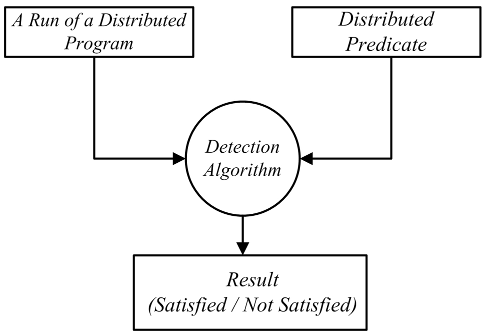
A Run of a Distributed Program
Distributed Predicate
Detection Algorithm
Result (Satisfied / Not Satisfied)

FIGURE 2. The runtime verification environment [17].

Step 2: Compute the best position globally reached by the swarm p_g (See the for loop shown in step 2 of Algorithm 1). $G()$ is a function that evaluates the goodness of each particle (How close it is to the optimal solution).

Step 3: The velocity of each particle is updated by (1):

$$v_{i,t+1} = v_{i,t} + C_1 r_1 (p_{i,t}^l - x_{i,t}) + C_2 r_2 (p_{g,t} - x_{i,t}). \quad (1)$$

where:

$t+1$ means the current time step, t means the previous time step.

$x_{i,t}$ the previous position of particle p_i.

$v_{i,t}$ is the previous velocity of particle p_i.

$v_{i,t+1}$ the new velocity of particle p_i.

$p_{i,t}^l$ is the best position reached by particle p_i in the previous time step.

Algorithm 1 Particle Swarm Optimization Algorithm

```
Step 1: (Initialization)
Randomly initialize $v_i$ and $x_i$ for each particle $p_i$
for $i = 1$ to Size of Swarm do
    $p_i^l = x_i$
end for

Step 2: (Compute $p_g$)
$p_g = x_1$
for $i = 2$ to Size of Swarm do
    if $G(x_i) > G(p_g)$ then
        $p_g = x_i$
    end if
end for

Step 3: (Update velocity)
$v_{i,t+1} = v_{i,t} + C_1 r_1 (p_{i,t}^l - x_{i,t}) + C_2 r_2 (p_{g,t} - x_{i,t})$

Step 4: (Update position)
$x_{i,t+1} = x_{i,t} + v_{i,t+1}$

Step 5: (Compute $p_i^l$ for each particle $p_i$)
for $i = 1$ to Size of Swarm do
    if $G(x_i) > G(p_i^l)$ then
        $p_i^l = x_i$
    end if
end for

Step 6: (Repeat Steps 2 to 5 until the satisfaction of the
termination criteria)
```

$p_{g,t}$ is the best position reached by the entire swarm in the previous time step.

C_1 and C_2 are the cognitive and the social parameters (usually $C_1 = C_2 = 2$).

r_1 and r_2 are two random numbers between 0 and 1.

Step 4: The position of each particle is updated by (2):

$$x_{i,t+1} = x_{i,t} + v_{i,t+1}. \tag{2}$$

where $x_{i,t+1}$ is the new position of particle p_i

Step 5: Update p_i^l for each particle p_i (See the for loop used for this purpose in step 5 of Algorithm 1).

Step 6: Repeat steps 2 to 5 until the satisfaction of the stopping criteria

In the next section, we will show in detail how we can exploit the PSO algorithm to design an algorithm for detecting distributed predicates.

V. PSO ALGORITHM FOR DETECTING DISTRIBUTED PREDICATES

In this paper, we will exploit the particle swarm optimization (PSO) algorithm in developing a distributed predicates detection (DPD) algorithm for predicates under the possibly modality. A distributed predicate under the possibly modality is said to be detected if there is at least one global state of the run in which the predicate is satisfied [13]. Its is not easy to adapt the PSO algorithm so that it can be used for solving the DPD problem. This is the main contribution of this research work.

Initially, we need to describe the input of the suggested detection algorithm. The DPD algorithm developed in this paper works on data collected at runtime. The collected data are stored in history files.

The distributed program will be instrumented so that processes will save in a history file the data necessary to detect the desired distributed predicate. As we have described in Section II, each process P_i executes a sequence of events $(e_{i0}, e_{i1}, \ldots)$, and generates a sequence of local states $(s_{i0}, s_{i1}, \ldots)$. The history file of a process contains several lines. Each line contains information about one local state. This information includes the values of the variables used in the predicate at the given local state and the vector clock timestamp of the event resulting in generating this local state. The general format of the history file is depicted in Figure 3. There are m lines in the file indicating that the process has generated m local states during the run under consideration. Each line has the form ***(k. <timestamp-k>, {(variable-1, value),…})***. Where ***k*** is the local state serial number, ***timestamp-k*** is the vector clock timestamp, and ***{(variable-1, value),…}*** is the set of the variables used in the desired predicate, along with its values at local state number ***k***.

```
Process ID: P_i
1. <timestamp-1>, { (variable-1, value), … }
•
•
•
k. <timestamp-k>, { (variable-1, value), … }
•
•
m. <timestamp-m>, { (variable-1, value), … }
```

FIGURE 3. The format of the history file generated by process P_i. Each line contains information about one local state of process P_i.

The following example demonstrates the above concepts. Given the run in Figure 1, and assuming that our goal is to detect the predicate $x + y + z = 9$. Each process must generate a history file at runtime containing the necessary information needed by the detection algorithm. For example, process P_1 should record, in the history file, information about the local states that could be part of a global state satisfying the predicate under consideration. Figure 4 shows the history files of the processes of the run. These files are the input to our detection algorithm. For example, the second line in the history file of process P_1 is (**2. <2,2,0>, {(y,10)}**). This line corresponds to the second local state of P_1, namely local state s_{11} in Figure 1. The vector clock timestamp of this local state is **<2,2,0>**, and the value of variable y of process P_1 at this local state is 10.

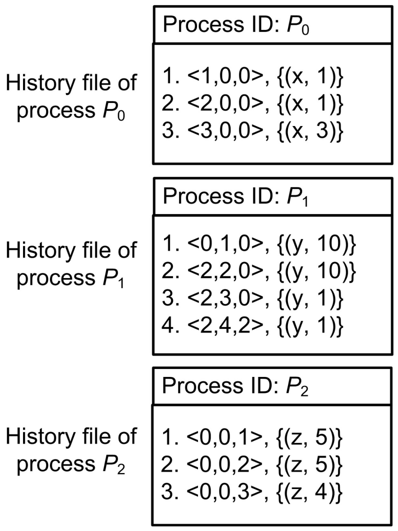

FIGURE 4. The set of history files generated by the run presented in Figure 1. These files contain the input necessary to the proposed distributed predicates detection algorithm.

In the following subsection, we will start with the detailed design of the proposed PSO-based distributed predicates detection algorithm.

A. THE DESIGN OF THE PSO-BASED DETECTION ALGORITHM

The first task in the design of the PSO-based DPD algorithm is to define the precise representation of a particle in our context. In general, a particle encodes a possible solution to the problem under consideration. In our PSO-based detection algorithm, a particle encodes a global state in which the predicate of interest may be evaluated to true. Because each local state has a serial number in the corresponding history file, we will use these numbers to represent each global state encoded by some particle. For example, Figure 5 shows a particle encoding the global state associated with the consistent cut depicted as a dashed line in the distributed run shown in Figure 1.

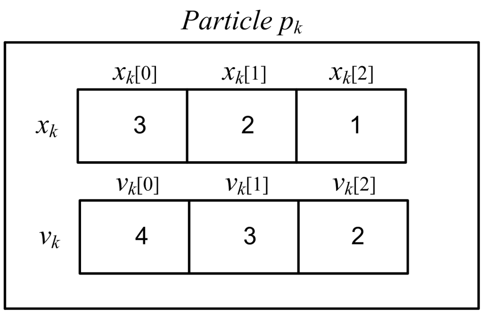

FIGURE 5. An example of a particle in the context of the PSO-based distributed predicates detection algorithm. This particle encodes the global state associated with the consistent cut depicted as a dashed line in Figure 1.

A particle p_k is an object involving two vectors of length equals to the number of processes involved in the predicate under consideration (3 processes in our simple example). The first vector x_k encodes the position of the particle and the second vector v_k encodes the velocity of the particle. The first element of the position of the particle p_k shown in Figure 5 ($x_k[0] = 3$) means that the third local state of P_0 is part of the global state encoded by this particle.

Assuming that the size of the swarm is s, the PSO-based DPD algorithm will initially generate s particles of the form shown in Figure 5. The position of each particle will be randomly initialized. After that, the algorithm needs to figure out the position of the particle that represents the best solution reached so far by the particles in the swarm (the global best position p_g). To accomplish this task, we need to precisely define the function $G()$ that will evaluate the goodness of a given particle p_k. The design of this function will be the next step in the design of our PSO-based detection algorithm.

Function $G()$ will mainly assign a value to a particle p_k indicating whether the global state encoded by this particle satisfies the intended predicate or not. If the particle satisfies the predicate, the goodness value assigned to it by the $G()$ function will be equal to 1. Otherwise, the particle goodness will be a value greater than or equal to 0 and less than 1 reflecting how close it is to a particle satisfying the predicate to be detected. A larger fitness value is better than a smaller one, because it indicates that the particle is closer to some global state satisfying the intended predicate.

To illustrate the $G()$ function, we will assume that our goal is to detect predicates of the form ($var_0 + \cdots + var_{n-1} = C$) in a run of a program involving n processes. var_i is a variable of P_i, and C is an integer constant. The $G()$ function will be as shown in equation (3). It will take the position of the particle to be evaluated as its input.

$$G(x_k) = \frac{1}{|(\sum_{i=0}^{n-1} value(i, x_k[i], var_i)) - C| + 1} \tag{3}$$

$G(x_k)$ is the goodness of particle p_k. $x_k[i]$ is the serial number of the local state of P_i involved in the global state encoded by particle p_k (See Figure 5). The function $value(i, x_k[i], var_i)$ returns the value of var_i of P_i in local state $x_k[i]$. This value can be found in the history file generated by process P_i.

For example, suppose we are going to check if the run in Figure 1 satisfies the distributed predicate $x+y+z = 9$. The goodness of the particle shown in Figure 5 will be evaluated according to equation (3) as follows:

$$G(x_k) = \frac{1}{|(\sum_{i=0}^{2} value(i,x_k[i],var_i))-9|+1}$$

$$G(x_k) = \frac{1}{|(value(0,x_k[0],x)+value(1,x_k[1],y)+value(2,x_k[2],z))-9|+1}$$

Using the particle shown in Figure 5, we can get the values of $x_k[0]$, $x_k[1]$, and $x_k[2]$. ($x_k[0] = 3$, $x_k[1] = 2$, and $x_k[2] = 1$)

$$G(x_k) = \frac{1}{|(value(0, 3, x)+value(1, 2, y) + value(2, 1, z))-9|+1}$$

Using the history files shown in Figure 4, we can get the values of $value(0, 3, x)$, $value(1, 2, y)$, and $value(2, 1, z)$.

For example, to get the value of $value(1, 2, y)$, we need to access the history file of process P_1 (see Figure 4), then we read local state number 2 to get the value of variable y at this local state. In our example, the value of y at local state 2 of process P_1 is 10. Similarly, the value of x at local state 3 of process P_0 is 3, and the value of z at local state 1 of process P_2 is 5.

$$G(p_k) = \frac{1}{|3+10+5-9|+1} = \frac{1}{10} = 0.1$$

Consequently, the goodness value of particle p_k shown in Figure 5 is (0.1).

One strong point of our PSO-based DPD algorithm is the fact that we do not need to develop an entire algorithm for different distributed predicates. If we have another distributed predicate to be detected, we just need few modifications on the $G()$ function. Other parts of the PSO-based detection algorithm will remain untouched.

The cut associated with a global state must be a consistent (see Section II). Hence, not any set of local states form a legitimate global state. However, since the particles are initialized randomly, some particles may not encode valid global states. To deal with this problem, there are several possible solutions. For example, we can initialize incorrect particles again randomly until we find a correct initialization. Another possible solution is to direct the $G()$ function to assign particles with invalid solutions a very low goodness value. This will result in ignoring their effect in subsequent iterations of the algorithm. We will adopt the second solution in our PSO-based detection algorithm. If the particle encodes an invalid global state, then the value assigned to it by the $G()$ function will be -1.

After describing the design of the $G()$ function, we can now describe the PSO-based detection algorithm in more details. The PSO-based detection algorithm will start by randomly initializing the particles in the swarm. The position of each particle will have a dimension equals to the number of processes in the run under consideration. Element i in the position of a particle corresponds to process P_i and will be randomly initialized with an integer number representing the serial number of one of P_i's local states.

In the next step, the PSO-based detection algorithm will initialize the local best position p_k^l for each particle p_k with an identical copy of the position of the same particle. The PSO-based detection algorithm will repeat the following steps until it finds a global state that satisfies the intended predicate or reaches the maximum number of iterations.

1) The detection algorithm uses the $G()$ function described above to evaluate the goodness of each particle in order to find the global best position p_g. The global best position is the position of the particle whose encoded global state is the nearest one in the swarm to some global state satisfying the predicate under consideration. (See step 2 of Algorithm 1)
2) Calculate the new velocity of each particle. We will use the equation shown in step 3 of Algorithm 1.
3) Update the position of each particle. We will use the equation shown in step 4 of Algorithm 1. Since the position of the particle in the context of our PSO-based detection algorithm consists of integer numbers only (representing local states numbers), we will round the value of the velocity before updating the position of the particle. Moreover, the new position may involve local states numbers outside the range of the local states of a given process. In this case, the new local state number is ignored and replaced with a randomly selected valid local state number.
4) Update the local best position for each particle (See step 5 of Algorithm 1)

An algorithm is sound if, anytime it returns an answer, the answer is true. Looking at the G() function that evaluates the goodness of a given particle, we can easily show that the algorithm is sound. The algorithm returns a solution (a global state where the value of the distributed predicate of interest is equal to true) if it found a global state whose goodness is 1. We are dealing with predicates of the form $(var_0 + \cdots + var_{n-1} = C)$. The goodness function $G()$ returns a goodness value of 1 only and only if $(var_0 + \cdots + var_{n-1} - C)$ equals to zero. And it will be equal to zero only if the intended predicate $(var_0 + \cdots + var_{n-1} = C)$ is true. As a result, if the algorithm reports that the value of a predicate is true ($G()$ returns 1), its answer will be true and hence the algorithm is sound.

In the next section, we will present and describe the experiments' results to evaluate the performance of the PSO-based DPD algorithm.

VI. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have used the Java programming language to implement the proposed PSO-based distributed predicates detection (DPD) algorithm. To evaluate its performance we used the PSO-based DPD algorithm to detect several distributed predicates of the form $(var_0 + \cdots + var_{n-1} = C)$ where n is the number of processes, var_i is an integer variable of P_i, and C is a constant. We have set the swarm size to 50, $C_1 = C_2 = 2$, and we have set that max number of iterations to 10^6.

We have conducted our experiments on a computer with Intel(R) Core(TM) i7-3770QM CPU, 3.4GHz with 8GB RAM. We have considered in our experiments different runs with different number of processes (50, 75, 100, 125, 150, 175, 200, 225, 250, 275 and 300 processes), and different number of events per process (5000, 10000, and 15000 events). In each scenario, we have executed the program implementing our PSO-based detection algorithm 200 times. We recorded the average execution time and the average number of iterations needed to detect a predicate of the form $(var_0 + \cdots + var_{n-1} = C)$. Table 1 summaries the experimental results. For example, given a run with 200 processes and 15000 events executed by each process, the PSO-based detection algorithm successfully detects the predicate $(var_0 + var_1 + \cdots + var_{199} = 2000)$ in 6845ms (on average).

TABLE 1. The experimental results of the PSO-based DPD algorithm.

Predicate	Number of Processes (n)	Number of Events executed by each process (m)	Time spent to detect the predicate/ms	Iterations
$var_0 + var_1 + \cdots + var_{49} = 500$	50	5000	2	16
		10000	2	16
		15000	3	17
$var_0 + var_1 + \cdots + var_{74} = 750$	75	5000	9	48
		10000	10	48
		15000	12	55
$var_0 + var_1 + \cdots + var_{99} = 1000$	100	5000	40	155
		10000	42	157
		15000	45	162
$var_0 + var_1 + \cdots + var_{124} = 1250$	125	5000	135	430
		10000	175	524
		15000	193	545
$var_0 + var_1 + \cdots + var_{149} = 1500$	150	5000	514	1365
		10000	548	1366
		15000	564	1384
$var_0 + var_1 + \cdots + var_{174} = 1750$	175	5000	1666	3658
		10000	1685	3532
		15000	2049	4183
$var_0 + var_1 + \cdots + var_{199} = 2000$	200	5000	6160	11466
		10000	6312	11471
		15000	6845	12130
$var_0 + var_1 + \cdots + var_{224} = 2250$	225	5000	17547	28712
		10000	19502	30253
		15000	21255	31612
$var_0 + var_1 + \cdots + var_{249} = 2500$	250	5000	54849	79406
		10000	65700	87560
		15000	68409	89480
$var_0 + var_1 + \cdots + var_{274} = 2750$	275	5000	171790	220488
		10000	228949	278299
		15000	258755	289577
$var_0 + var_1 + \cdots + var_{299} = 3000$	300	5000	584557	665015
		10000	674581	730073
		15000	831917	895353

The detection of the predicate ($var_0 + \cdots + var_{n-1} = C$) is an NP-complete problem in general [36]. Consequently, there is no polynomial time algorithm for detecting predicates of this form. The only way to detect it is to go through the global states of the run until we find a global state satisfying the predicate. This detection process will be time-consuming and requires exploring up to $O(m^n)$ global states in the worst case, where n is the number of processes, and m is the number of local states at any process (See Section II). For example, in a run with five processes and 400 local states for each process, we need more than 2.8 hours to detect the predicate in the worst case, assuming that we can check 10^9 global states every second. Consequently, the developed PSO-based DPD algorithm has a better performance.

In [36], the authors have developed efficient algorithm to detect distributed predicates of the form ($var_0 + \cdots + var_{n-1} = C$) under some restricted conditions. Specifically, their algorithm works efficiently assuming that the predicates variables are incremented or decremented by at most one at each step. We have extended our experiments to evaluate the performance of the PSO-Based DPD algorithm in detecting distributed predicates under the specified restrictions. We have updated our programs to meet the restrictions imposed on the predicate variables and we have performed several experiments to compare the results. In fact, the G() function is the only part of the PSO-Based DPD algorithm that has to be modified, the remaining parts need not be changed. This reflects the flexibility of the proposed DPD algorithm.

The results shown in Table 2 demonstrate that the performance of our PSO-Based detection algorithm in detecting distributed predicates with some restriction has been significantly improved. We have considered in our experiments different runs involving (50, 100, 150, 200, 250, 300, 350, and 400 processes). In each scenario, we have executed the program implementing our PSO-based detection algorithm 200 times, and we have calculated the average execution time and the average number of iterations needed to detect a predicate. For example, given a run with 300 processes and 15000 events executed by each process, the PSO-based detection algorithm successfully detects the predicate ($var_0 + var_1 + \cdots + var_{299} = 300$) in 84ms (on average) assuming

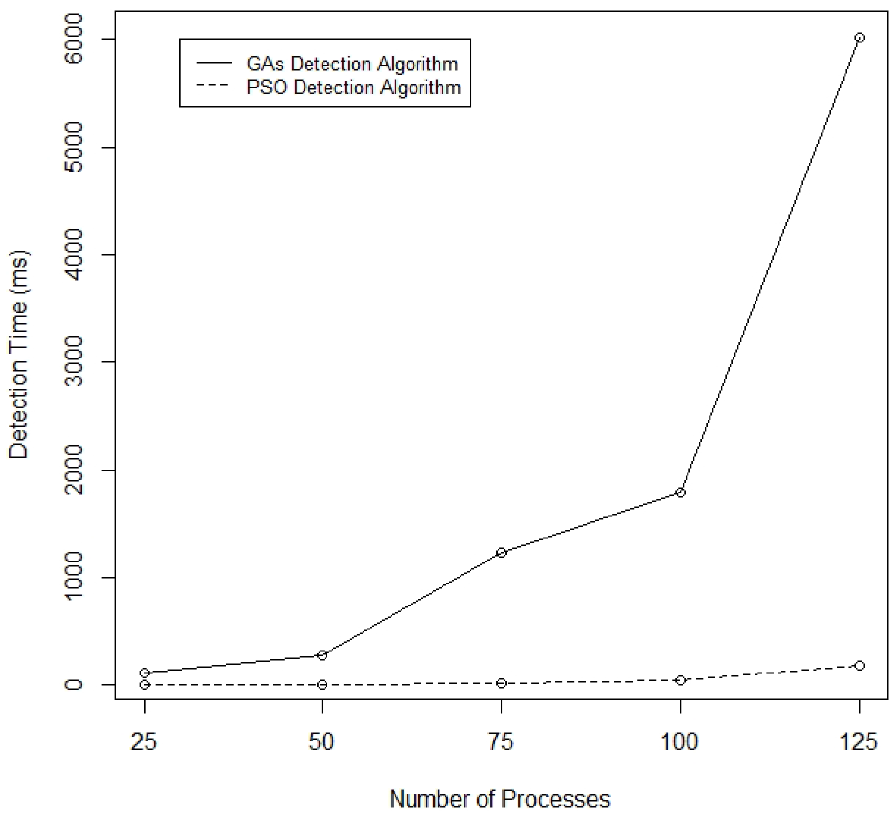

FIGURE 6. The performance of the PSO-based detection algorithm compared with the performance of the detection approach based on Genetic Algorithms.

TABLE 2. The performance of the PSO-based DPD algorithm in detecting distributed predicates of the form ($var_0 + \cdots + var_{n-1} = c$) assuming that the predicates variables are incremented or decremented by at most one at each step.

Predicate	Number of Processes (n)	Number of Events executed by each process (m)	Time spent to detect the predicate/ms	Iterations
$var_0 + var_1 + \cdots + var_{49} = 50$	50	15000	2	13
$var_0 + var_1 + \cdots + var_{99} = 100$	100		6	19
$var_0 + var_1 + \cdots + var_{149} = 150$	150		16	34
$var_0 + var_1 + \cdots + var_{199} = 200$	200		24	38
$var_0 + var_1 + \cdots + var_{249} = 250$	250		63	85
$var_0 + var_1 + \cdots + var_{299} = 300$	300		84	93
$var_0 + var_1 + \cdots + var_{349} = 350$	350		130	109
$var_0 + var_1 + \cdots + var_{399} = 400$	400		332	232

that the predicates variables are incremented or decremented by at most one at each step.

We have extended our experiments to evaluate the performance of the proposed PSO-based DPD algorithm on detecting conjunctive distributed predicates of the form ($t_0 \wedge t_1 \wedge \cdots \wedge t_{n-1}$) where term t_i is a local predicate of process P_i (involves only variables of process P_i).

To enable the PSO-based DPD algorithm to deal with such kinds of predicates, the only thing that we need to modify is the $G()$ function, other parts of the algorithm will remain untouched. This is one of the advantages of the proposed PSO-based DPD algorithm compared with other detection algorithms that work efficiently for specific kinds of predicates only. We have modified the $G()$ function so that it will return the number of terms (local predicates) in the conjunctive predicate that evaluates to false. A conjunctive predicate is evaluated to true in a given global state if the $G()$ function returns zero, which means that all of the terms $t_0, t_1, \ldots, t_{n-1}$ in the conjunctive predicate are true, and hence, $t_0 \wedge t_1 \wedge \cdots \wedge t_{n-1}$ is also true.

We have tested our algorithm in detecting conjunctive predicates of the form ($x_0 > c_0 \wedge \cdots \wedge x_{n-1} > c_{n-1}$) where x_i is a variable of process P_i and $c_0, \ldots, c_{n-1}$ are constants. Table 3 demonstrates the results of the experiments performed to evaluate the PSO-based DPD algorithms on several conjunctive predicates. We have considered in our

TABLE 3. The performance of the PSO-based DPD algorithm in detecting distributed predicates of the form $(x_0 > c_0 \wedge x_1 > c_1 \wedge \cdots \wedge x_{n-1} > c_{n-1})$ where x_i is a variable of process P_i and $c_0, \ldots, c_{n-1}$ are constants.

Predicate	Number of Processes (n)	Number of Events executed by each process (m)	Time spent to detect the predicate/ms	Iterations
$(x_0 > 10 \wedge x_1 > 10 \wedge \cdots \wedge x_{199} > 10)$	200	15000	2	1
$(x_0 > 10 \wedge x_1 > 10 \wedge \cdots \wedge x_{299} > 10)$	300		18	15
$(x_0 > 10 \wedge x_1 > 10 \wedge \cdots \wedge x_{399} > 10)$	400		199	147
$(x_0 > 10 \wedge x_1 > 10 \wedge \cdots \wedge x_{499} > 10)$	500		2469	1525
$(x_0 > 10 \wedge x_1 > 10 \wedge \cdots \wedge x_{599} > 10)$	600		22153	11446

experiments different runs with different number of processes (200, 300, 400, 500, and 600 processes). In each scenario, we have executed the program implementing our PSO-based detection algorithm 200 times, and we have recorded the average execution time and the average number of iterations needed to detect a predicate. It is clear from the results that the proposed algorithm can efficiently detect conjunctive predicates for distributed programs runs with several hundreds of processes. For example, the PSO-based DPD algorithm detects the predicate $(x_0 > 10 \wedge x_1 > 10 \wedge \cdots \wedge x_{499} > 10)$ in 2469 ms.

We have implemented the genetic algorithms detection approach presented in [33], and we have conducted several experiments to compare its performance with the performance of the PSO-based detection algorithm presented in this paper. Figure 6 shows that the PSO-based distributed predicates detection algorithm clearly outperforms the performance of the genetic algorithms detection approach [33]. The results in [32], [33] demonstrate that the genetic algorithms detection approach outperforms the harmony algorithm detection approach. Consequently, our proposed PSO-based detection algorithm also outperforms the harmony algorithm distributed predicates detection approach.

The branch and bound ($B\delta B$) and the A^* search algorithms are two well-known algorithms that could be exploited in solving the DPD algorithm. However, the biggest drawback of the $B\delta B$ and the A^* search algorithms is the exponential space complexity. The A^* algorithm keeps all generated nodes in memory (to avoid visiting the same node multiple times) and the $B\delta B$ algorithm keeps the branch tree in memory. Consequently, when solving problems with exponential search space size, like the DPD problem, you can run out of RAM before finding the optimal solution. In future research papers, we will consider developing a hybrid detection approach in which metaheuristic algorithms are combined with A^* and $B\delta B$.

VII. CONCLUSION AND FUTURE WORK

The detection of distributed predicates is a highly challenging problem with practical applications, including context change detection in pervasive environments, testing, debugging, and monitoring distributed programs. This paper presents a distributed predicates detection (DPD) algorithm inspired by the particle swarm optimization (PSO) algorithm to detect distributed predicates under the possibly modality efficiently. The experimental results confirm that the proposed PSO-based DPD algorithm outperforms the traditional enumerated based DPD algorithm.

In our experiments, we have studied distributed predicates of the form $(var_0 + \cdots + var_{n-1} = C)$. The current work can be extended by considering the design and the implementation of the $G()$ function for other forms of distributed predicates. Several other experiments can be conducted, and some parameter tuning techniques can be exploited to find the best value for each parameter of the PSO-based detection algorithm to improve its performance. We can also consider solving the DPD problem using other known metaheuristic algorithms like monarch butterfly optimization (MBO) [41], earthworm optimization algorithm (EWA) [42], elephant herding optimization (EHO) [43], moth search (MS) algorithm [44], slime mould algorithm (SMA) [45], and harris hawks optimization (HHO) [46]. Moreover, we can consider the development of detection algorithms for distributed predicates under fine-grained modalities [47], [48].

REFERENCES

[1] K. V. Garg and R. Garg, "Parallel algorithms for predicate detection," in *Proc. 20th Int. Conf. Distrib. Comput. Netw. (ICDCN)*, New York, NY, USA, 2019, pp. 51–60.

[2] K. V. Garg, "Applying predicate detection to the constrained optimization problems," *CoRR*, vol. abs/1812.10431, Dec. 2018.

[3] E. A. Maghayreh, "An artificial bee colony algorithm for detecting distributed predicates," *Int. J. Distrib. Syst. Technol.*, vol. 9, no. 3, pp. 53–64, Jul. 2018.

[4] S. Yingchareonthawornchai, V. T. Valapil, S. Kulkarni, E. Torng, and M. Demirbas, "Efficient algorithms for predicate detection using hybrid logical clocks," in *Proc. 18th Int. Conf. Distrib. Comput. Netw. (ICDCN)*, New York, NY, USA, 2017, pp. 10:1–10:10.

[5] A. Natarajan, H. Chauhan, N. Mittal, and K. V. Garg, "Efficient abstraction algorithms for predicate detection," *Theor. Comput. Sci.*, vol. 688, pp. 24–48, Nov. 2017.

[6] Y. Yang, Y. Huang, X. Ma, and J. Lu, "Enabling context-awareness by predicate detection in asynchronous environments," *IEEE Trans. Comput.*, vol. 65, no. 2, pp. 522–534, Feb. 2016.

[7] Y.-J. Chang and K. V. Garg, "Predicate detection for parallel computations with locking constraints," in *Proc. 20th Int. Conf. Princ. Distrib. Syst. (OPODIS)*, 2016, pp. 17:1–17:17.

[8] W. Zhu, J. Cao, and M. Raynal, "Predicate detection in asynchronous distributed systems: A probabilistic approach," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 173–186, Jan. 2016.

[9] M. Shen and A. D. Kshemkalyani, "Hierarchical detection of strong unstable conjunctive predicates in large-scale systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2899–2908, Nov. 2014.

[10] H. Chauhan, V. K. Garg, A. Natarajan, and N. Mittal, "A distributed abstraction algorithm for online predicate detection," in *Proc. IEEE 32nd Int. Symp. Reliable Distrib. Syst.*, Sep. 2013, pp. 101–110.

[11] A. D. Kshemkalyani and J. Cao, "Predicate detection in asynchronous pervasive environments," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1823–1836, Sep. 2013.

[12] M. Shen, A. D. Kshemkalyani, and A. Khokhar, "Detecting tree distributed predicates," in *Proc. 41st Int. Conf. Parallel Process. Workshops*, Sep. 2012, pp. 598–599.

[13] M. C. Chase and K. V. Garg, "Detection of global predicates: Techniques and their limitations," *Distrib. Comput.*, vol. 11, no. 4, pp. 191–201, 1998.

[14] Y. Huang, X. Ma, J. Cao, X. Tao, and J. Lu, "Concurrent event detection for asynchronous consistency checking of pervasive context," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun.*, 2009, pp. 1–9.

[15] D. Garlan, D. P. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: Toward distraction-free pervasive computing," *IEEE Pervas. Comput.*, vol. 1, no. 2, pp. 22–31, Apr. 2002.

[16] Y. Yang, Y. Huang, J. Cao, X. Ma, and J. Lu, "Formal specification and runtime detection of dynamic properties in asynchronous pervasive computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1546–1555, Aug. 2013.

[17] E. A. Maghayreh, *Simplifying Runtime Verification of Distributed Programs: Ameliorating the State Space Explosion Problem*. Berlin, Germany: VDM Verlag, 2010.

[18] Z. Li, L. Qiu, R. Li, Z. He, J. Xiao, Y. Liang, F. Wang, and J. Pan, "Enhancing BCI-based emotion recognition using an improved particle swarm optimization for feature selection," *Sensors*, vol. 20, no. 11, pp. 1–16, 2020.

[19] M. Van, D. T. Hoang, and H. J. Kang, "Bearing fault diagnosis using a particle swarm optimization-least squares wavelet support vector machine classifier," *Sensors*, vol. 20, no. 12, pp. 1–19, 2020.

[20] A. Naz, N. Javaid, T. N. Qureshi, M. Imran, M. Ali, and Z. A. Khan, "EDHBPSO: Enhanced differential harmony binary particle swarm optimization for demand side management in smart grid," in *Proc. 32nd Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, May 2018, pp. 218–225.

[21] H. Xing and X. Pan, "Application of improved particle swarm optimization in system identification," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Jun. 2018, pp. 1341–1346.

[22] W. Yu, Z. Zeng, B. Peng, S. Yan, Y. Huang, H. Jiang, X. Li, and T. Fan, "Multi-objective optimum design of high-speed backplane connector using particle swarm optimization," *IEEE Access*, vol. 6, pp. 35182–35193, 2018.

[23] T. Kaur and D. Kumar, "Particle swarm optimization-based unequal and fault tolerant clustering protocol for wireless sensor networks," *IEEE Sensors J.*, vol. 18, no. 11, pp. 4614–4622, Jun. 2018.

[24] S. B. Sakri, N. B. A. Rashid, and Z. M. Zain, "Particle swarm optimization feature selection for breast cancer recurrence prediction," *IEEE Access*, vol. 6, pp. 29637–29647, 2018.

[25] R. Hassan, B. Cohanim, O. de Weck, and G. Venter, "A comparison of particle swarm optimization and the genetic algorithm," in *Proc. 46th AIAA/ASME/ASCE/AHS/ASC Struct., Struct. Dyn. Mater. Conf.*, Apr. 2005. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2005-1897

[26] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[27] F. Mattern, "Virtual time and global states of distributed systems," in *Proc. Int. Workshop Parallel Distrib. Algorithms*, Château de Bonas, France, Oct. 1989, pp. 215–226.

[28] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 1, pp. 63–75, 1985.

[29] M. Spezialetti and P. Kearns, "Efficient distributed snapshots," in *Proc. ICDCS*, 1986, pp. 382–388.

[30] V. K. Garg and B. Waldecker, "Detection of strong unstable predicates in distributed programs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 12, pp. 1323–1333, Dec. 1996.

[31] E. A. Maghayreh, "A simulated annealing based algorithm for detecting distributed predicates," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2017, pp. 1612–1617.

[32] E. A. Maghayreh, "A harmony search based algorithm for detecting distributed predicates," *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 10, pp. 153–160, 2012.

[33] E. A. Maghayreh, I. A. Doush, and F. Alkhateeb, "Detecting distributed predicates using genetic algorithms," *Int. J. Intell. Inf. Technol.*, vol. 9, no. 1, pp. 56–70, 2013.

[34] N. Mittal, A. Sen, and V. K. Garg, "Solving computation slicing using predicate detection," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 12, pp. 1700–1713, Dec. 2007.

[35] N. Mittal and K. V. Garg, "Techniques and applications of computation slicing," *Distrib. Comput.*, vol. 17, no. 3, pp. 251–277, Mar. 2005.

[36] N. Mittal and V. K. Garg, "On detecting global predicates in distributed computations," in *Proc. 21st Int. Conf. Distrib. Comput. Syst.*, Apr. 2001, pp. 3–10.

[37] E. A. Maghayreh, "Block-based atomicity to simplify the verification of distributed applications," in *Proc. 24th Can. Conf. Electr. Comput. Eng.*, 2011, pp. 887–891.

[38] H. F. Li and E. A. Maghayreh, "Checking distributed programs with partially ordered atoms," in *Proc. 14th Asia–Pacific Softw. Eng. Conf. (APSEC)*, 2007, pp. 518–525.

[39] H. F. Li and E. A. Maghayreh, "Using synchronized atoms to check distributed programs," in *Proc. 13th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Hsinchu, Taiwan, 2007, pp. 1–8.

[40] J. Kennedy and C. R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Nov. 1995, pp. 1942–1948.

[41] G.-G. Wang, S. Deb, and Z. Cui, "Monarch butterfly optimization," *Neural Comput. Appl.*, vol. 31, pp. 1995–2014, May 2015.

[42] G.-G. Wang, S. Deb, and L. dos S. Coelho, "Earthworm optimisation algorithm: A bio-inspired metaheuristic algorithm for global optimisation problems," *Int. J. Bio-Inspired Comput.*, vol. 12, no. 1, pp. 1–22, Jan. 2018.

[43] G. Wang, S. Deb, and L. D. S. Coelho, "Elephant herding optimization," in *Proc. 3rd Int. Symp. Comput. Bus. Intell. (ISCBI)*, 2015, pp. 1–5.

[44] G.-G. Wang, "Moth search algorithm: A bio-inspired metaheuristic algorithm for global optimization problems," *Memetic Comput.*, vol. 10, pp. 151–164, Jun. 2018.

[45] S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, "Slime mould algorithm: A new method for stochastic optimization," *Future Gener. Comput. Syst.*, vol. 111, pp. 300–323, Oct. 2020.

[46] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Gener. Comput. Syst.*, vol. 97, pp. 849–872, Aug. 2019.

[47] P. Chandra and A. D. Kshemkalyani, "Causality-based predicate detection across space and time," *IEEE Trans. Comput.*, vol. 54, no. 11, pp. 1438–1453, Nov. 2005.

[48] A. D. Kshemkalyani, "A fine-grained modality classification for global predicates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 8, pp. 807–816, Aug. 2003.

ESLAM AL MAGHAYREH (Member, IEEE) received the B.S. and M.S. degrees in computer science from Yarmouk University, Irbid, Jordan, in 2001 and 2003, respectively, and the Ph.D. degree in computer science from Concordia University, Montreal, Canada, in 2008.

In 2008, he joined the Department of Computer Science, Yarmouk University, as an Assistant Professor. Since 2014, he has been an Associate Professor with the College of Applied Computer Science, King Saud University, Riyadh, Saudi Arabia. He has authored/edited three books and more than 30 articles. His research interests include distributed systems, computational intelligence, runtime verification of distributed programs machine learning, and data science.

HABIB DHAHIRI was born in Sidi Bouzid, Tunisia, in 1975. He received the B.S. and Ph.D. degrees in computer science from the National Engineering School of Sfax (ENIS), Tunisia, in 2001 and 2013, respectively. Since 2015, he has been an Assistant Professor with the College of Applied Computer Science, King Saud University, Riyadh, Saudi Arabia. He has authored and coauthored more than 17 publications in journals and conferences. His research interests include computational intelligence, neural networks, swarm intelligence, differential evolution, and genetic algorithms.

FAHAD ALBOGAMY received the B.Sc. degree in information systems (Hons.) from King Saud University, in 2003, and the M.Sc. and Ph.D. degrees (Hons.) in computer sciences from The University of Manchester, U.K., in 2010 and 2017, respectively. He was the First Dean of the Applied Computer Sciences College, King Saud University. He is currently an Assistant Professor with Taif University, Saudi Arabia. He is interested in artificial intelligence, big data, machine learning, NLP, and digital image processing.

MOHAMAD MAHMOUD AL RAHHAL (Member, IEEE) received the B.Sc. degree in computer engineering from Aleppo University, Aleppo, Syria, in 2002, the M.Sc. degree from Jamia Hamdard University, New Delhi, India, in 2005, and the Ph.D. degree in computer engineering from King Saud University, Riyadh, Saudi Arabia, in 2015. From 2006 to 2012, he was a Lecturer with Al Jouf University, Sakakah, Saudi Arabia. Since 2015, he has been an Assistant Professor in computer science with King Saud University. His research interests include signal/image medical analysis, remote sensing, and computer vision.

AWAIS MAHMOOD received the B.S. and M.S. degrees in computer engineering from Eastern Mediterranean University, TRNC, Turkey, in 2001 and 2003, respectively, and the Ph.D. degree in computer engineering from King Saud University, Riyadh, Saudi Arabia, in 2014. He is currently an Assistant Professor with the College of Applied Computer Science, King Saud University. His research interests include image and speech processing, signal processing for healthcare, data science and big data processing, and software systems.

ESAM OTHMAN received the B.Sc. (Hons.) degree in computer engineering from Umm Al-Qura University, Mecca, Saudi Arabia, in 2007, and the M.Sc. and Ph.D. degrees in computer engineering from King Saud University, Riyadh, Saudi Arabia, in 2012 and 2016, respectively. He is currently working as an Assistant Professor with the College of Applied Computer Sciences, King Saud University. His research interests include machine learning, pattern recognition, and remote sensing.

WAIL S. ELKILANI received the M.Sc. and Ph.D. degrees from the Faculty of Engineering, Cairo University, Egypt, in 1996 and 2001 respectively.

He has held a postdoctoral position with the Advanced Networking Laboratory, The University of British Columbia, Vancouver, Canada, in 2006. Since 2015, he has been a Full Professor with the College of Applied Computer Science, King Saud University, Riyadh, Saudi Arabia. He has supervised nearly 20 master and Ph.D. students. He has authored over 30 articles on networking and computer systems in international journals and conferences. His research interests include *ad-hoc* wireless networks, internet protocols and systems performance evaluation of parallel and networked computer systems. Lately, he is working in exploring hacking threats in wired LANs, WLANs, and VoIP networks. He was chosen as a member of the Cisco Advisory Board for the year 2009/2010. He was awarded by Cisco Systems as the Best Instructor of Cisco Academies in 2006 and 2008, for his innovation in integrating Cisco curriculums with academic networking courses.