


# Adaptive pursuit learning for energy-efficient target coverage in wireless sensor networks

Ramesh Upreti<sup>1</sup> | Ashish Rauniyar<sup>1,2</sup> | Jeevan Kunwar<sup>1</sup> | Hårek Haugerud<sup>1</sup> |  
Paal Engelstad<sup>1,2</sup> | Anis Yazidi<sup>1</sup> 

<sup>1</sup>Department of Computer Science, Oslo Metropolitan University, Oslo, Norway

<sup>2</sup>Department of Technology Systems, University of Oslo, Oslo, Norway

## Correspondence

Anis Yazidi, Department of Computer Science, Oslo Metropolitan University, Pilestredet 52, Oslo 0167, Norway.  
Email: anisy@oslomet.no

## Summary

With the proliferation of technologies such as wireless sensor networks (WSNs) and the Internet of things (IoT), we are moving towards the era of automation without any human intervention. Sensors are the principal components of the WSNs that bring the idea of IoT into reality. Over the last decade, WSNs are being used in many application fields such as target coverage, battlefield surveillance, home security, health care monitoring, and so on. However, the energy efficiency of the sensor nodes in WSN remains a challenging issue due to the use of a small battery. Moreover, replacing the batteries of the sensor nodes deployed in a hostile environment frequently is not a feasible option. Therefore, intelligent scheduling of the sensor nodes for optimizing its energy-efficient operation and thereby extending the life-time of WSN has received a lot of research attention lately. In particular, this article investigates extending the lifetime of the WSN in the context of target coverage problems. To tackle this problem, we propose a scheduling technique for WSN based on a novel concept within the theory of learning automata (LA) called pursuit LA. Each sensor node in the WSN is equipped with an LA so that it can autonomously select its proper state, that is, either sleep or active, with an aim to cover all targets with the lowest energy cost possible. Our comprehensive experimental testing of the proposed algorithm not only verifies the efficiency of our algorithm, but it also demonstrates its ability to yield a near-optimal solution. The results are promising, given the low computational footprint of the algorithm.

## KEYWORDS

adaptive pursuit learning, energy efficiency, learning automata, minimum active sensors set, target coverage, wireless sensor network

## 1 | INTRODUCTION

With the explosive development of the Internet of things (IoT) technologies, various applications have surged up such as smart city, smart homes, smart hospitals, smart transportation, and so on.<sup>1,2</sup> We are witnessing the era of complete automation without any human intervention. All these applications of IoT are possible only because of the use of sensors. Sensors are the principal components that bring the idea of IoT into reality.<sup>3</sup> However, these sensors are an integral part of the wireless sensor networks (WSNs). The United States military first developed WSN technology as a sound surveillance system to detect and track submarines in the 1950s.<sup>4</sup> Since then WSN has gone through a vast amount of transitions and

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2020 The Authors. *Concurrency and Computation: Practice and Experience* published by John Wiley & Sons, Ltd.

thus it has become a practical solution for many applications such as target tracking,<sup>5</sup> smart factory,<sup>6</sup> surveillance,<sup>7</sup> target coverage,<sup>8</sup> industrial automation,<sup>9</sup> and so on. A WSN network comprises of the vast amount of sensors which brings together the sensed data of various activities of the physical world to give us meaningful information.<sup>10</sup> It should be noted that these sensor nodes in a WSN network operate autonomously with small batteries, which lasts from a number of months to years. Recharging or replacing the batteries of these sensor nodes is not a feasible option, especially if it is deployed in a hostile environment such as chemical reactors, underground tunnels, nuclear plants, and so on.<sup>11</sup> Therefore, the energy efficiency of these sensor nodes in WSN is a major concern and has attracted a great deal of attention from academia to industry. In this regard, the primary intention of this article is to expand the network life-time of the whole WSN system. One of the possible ways through which network life-time could be enhanced is by effectively covering the target in an energy-efficient way. The network life-time is defined in the context of network coverage problems as the duration of time elapsed from the network starts functioning with full coverage from its initialization to the time instant where the coveted coverage criteria is unsatisfied.<sup>8</sup>

Target coverage is one of the important areas of the WSNs. It has been widely used for monitoring purposes, and it has its main application in military surveillance.<sup>12</sup> This coverage area can be defined as the area within which a sensor node can track the activities of the specified target. The mobile sensor deployment problem and the target coverage problem in mobile WSNs are NP-Hard.<sup>13</sup> The target coverage problem by the sensor nodes in the WSN includes three families of problems which are defined as follows according to Reference 14:

- **Area coverage:**

This coverage problem is concerned with the monitoring of the targets in the entire area of the network.

- **Target coverage:**

This coverage problem is concerned with the monitoring of only certain targets within the specified region of the network.

- **Barrier coverage:**

The barrier coverage problem aims at minimizing the probability of undetected penetration through the barrier in the network.

The sensor node has a sensing area coverage based on its sensing range. Moreover, the sensor node also has a radio area coverage based on its communication range.<sup>15</sup> Intuitively, the specified monitoring field consists of densely deployed sensor nodes to avoid the formation of the coverage holes in the network. Particularly for the harsh terrains which are difficult to access, the sensor nodes are deployed in a random manner using an aircraft. When deployed randomly, usually, the areas covered by these sensor nodes overlap with each other. Due to the dense random deployment of the sensor nodes, there might be the sensing area that completely overlaps with the sensing area of other redundant sensor nodes. In WSNs, it is not advised to have multiple redundant sensors to cover the same target or area if it is already covered by other sensor nodes. Covering the same target or area by the redundant sensors affect the network-lifetime to a greater extent. Therefore, there is a multitude of research addressing the issue of identifying the redundant sensors and intelligently scheduling them in distinct time slots. Usually scheduling is done in such a way that a sensor can alternate between "Active" and "Sleep" mode to meet desired coverage requirements.<sup>16</sup>

## 2 | RELATED WORKS AND CONTRIBUTIONS

A comprehensive survey on energy-efficient coverage protocols in WSNs was carried out in References 15,17. In Reference 12, an energy-efficient scheduling algorithm called  $EC^2$  was proposed to address the problems of energy-efficiency, coverage, and connectivity in military surveillance applications using the fuzzy graphs. An adaptive energy-efficient target detection based on mobile WSNs was proposed in Reference 18. The authors proposed a clustering algorithm based on k-means++, and an energy control mechanism was used to reduce the energy consumption of nodes. A genetic algorithm (GA) approach for k-coverage and m-connected node placement in target based WSNs was proposed in Reference 19. In order to fulfill both coverage and connectivity requirements, the authors' scheme found a minimum number of potential positions to place the sensor nodes for a given set of target points. The detailed working of GA is presented in Reference 20. In Reference 21, node failure and battery-aware coverage protocol called optimized discharge-curve-based coverage protocol (ODCP) for WSNs was proposed. The authors' ODCP protocol used battery discharge rate, failure probability, and coverage overlap information to determine optimal sleep schedules for redundant nodes and thereby reducing the energy consumption of the network.

Furthermore, an energy-efficient distributed algorithm for weak-barrier coverage using event-based clustering and adaptive sensor rotation was proposed in Reference 22. In Reference 23, a system using the GA to optimize k-coverage criteria in WSNs to ensure continuous monitoring of targets with limited energy resources for the longest possible time was considered. The authors' proposed model extends the network lifetime in a range of 23% to 45.7%. Moreover, through the proposed algorithm, the authors also showed that the running time to form the network structure was reduced by 12%. A minimum energy target tracking with coverage guarantee in WSNs was proposed in Reference 24. The authors showed that the residual capacities of the sensors could be preserved and balanced to carry out additional target tracking missions using the same WSN. Some WSN monitoring applications may require that at least k directional nodes in m-connected WSNs track the targets. Therefore, the authors in Reference 25

proposed a target tracking system using directional nodes. Network coding is an effective method used to improve network throughput and, thus, efficiency by combining the packets at the intermediate nodes from multiple sources, thereby reducing the number of transmissions. Therefore, the authors in Reference 26, proposed a novel approach for target coverage in WSN based on network coding technique.

While the methods described in the above literature address the problem of target coverage in WSN. But at the same time, these methods are complex and hence are not scalable. One of the scheduling methods for energy-efficient WSN is through learning automata (LA).<sup>14,27</sup> This LA provides the sensor node with the purpose of learning their state and selecting their correct state, that is, either active or sleep mode, to extend the lifetime of the WSN's network.<sup>16</sup>

In Reference 28, a pDCDS algorithm was proposed to find a minimum number of nodes to monitor  $p$ -percent coverage in the deployed network. The authors investigated connected  $p$ -percent problem in WSNs and devised a degree-constrained minimum-weight connected dominating set to solve the coverage problem. The authors' pDCDS algorithm used LA to select the best dominator and dominate sensor nodes to cover  $p$ -percent coverage in the networks. Reference 29 proposed a topology control protocol based on LA, which is named as LBLATC. Using distributed LA, the authors reduced the energy consumption by dynamically optimizing the transmission range while keeping the connectivity. In Reference 30, a new distributed approach using LA scheme was proposed to maximize the number of barriers and minimize the energy consumption for the border surveillance in WSNs. The authors proposed a distributed border surveillance (DBS) algorithm aimed to find the minimum possible number of nodes in each barrier to monitor the network borders. In the DBS approach, a LA was used to find the best nodes to assure barrier coverage at any moment. In Reference 31, the authors proposed an imperialist competitive algorithm-based method called ICABC to solve the barrier coverage problems in WSN. ICABC algorithm managed to find the best sensor nodes to cover the network barriers in the deployed network. In Reference 32, the authors focused on the problem of partial coverage, targeting scenarios in which the continuous monitoring of a limited portion of the area of interest is enough. The authors presented a novel algorithm called partial coverage learning automata that relies on LA to implement sleep scheduling approaches to minimize the number of active sensors for covering the desired portion of the region of interest, preserving the connectivity among sensors. Similarly, in Reference 33, the authors proposed a greedy heuristic algorithm to deal with the coverage problem of heterogeneous WSNs in the case when the complete coverage of the network is not needed. In Reference 16, an energy-efficient scheduling algorithm based on LA for the target coverage problem was proposed. In this scheme, a sensor node selected its appropriate state (either active or sleep) with the help of LA-based technique.

A LA-based scheduling algorithm was designed in Reference 34 to find an optimal solution to the problem of target coverage, which can generate both disjoint and nondisjoint cover sets within the WSNs. In this scheme, one LA is responsible for selecting the sensor nodes that should be enabled at each point to cover all the targets. This is similar to pursuit learning. They use a dynamic threshold to determine a reward. The reward is granted whenever the size of the current set of active sensors is reduced. However, this will lead to slow convergence. As time proceeds, it will even be harder to get better solutions (lower the threshold), and the reward probability will decrease. In Reference 35, the authors extend the previous work in Reference 34 with an algorithm where they designed a pruning rule to manage critical targets in order to maximize network lifetime. A pruning rule was specifically designed to avoid the selection of redundant sensors. The authors also demonstrated that selecting the learning rate with a high accuracy leads to a longer network lifetime. A LA-based algorithm for solving coverage problem in directional sensor networks was proposed in Reference 36. The authors proposed a fully distributed algorithm based on irregular cellular LA to find a near-optimal solution for selecting the correct working path for each sensor node. Then, a centralized approximation algorithm based on distributed LA was proposed that can cover all targets with the minimum number of active sensors. The proposed algorithm is a fully distributed algorithm in which each sensor node chooses its optimal direction solely based on the directions selected and the targets covered by its adjacent sensor nodes. Different from other works,<sup>37</sup> tackled the issue of having maximum network lifetime with adjustable ranges (MNLARs) and proposed a heuristic MNLAR algorithm for directional sensor networks. In situations where the sensors have several directions and different sensing ranges, the authors suggested two greedy-based scheduling algorithms aimed at selecting the correct sensor directions and sensing ranges so as to meet the target coverage problem requirement and, at the same time, optimize the lifetime of the network. Similar to Reference 37, a new LA-based approach equipped with a pruning rule for maximizing network lifetime in WSNs with adjustable sensing ranges was proposed in Reference 38. Furthermore, a new genetic-based approach for maximizing network lifetime in directional sensor networks with adjustable sensing ranges was proposed in Reference 39. The authors suggested a target-oriented GA-based algorithm that could form cover sets consisting of sensors with suitable directions and sensing ranges so that all network targets could be tracked desirably. In Reference 40, the authors explored the priority-based target coverage issue where the targets have different requirements for coverage quality, and as a result, they may need to be tracked by more than one direction of sensors. In order to solve this problem, they suggested an LA-based scheduling algorithm to find a minimum subset of sensor directions to track and satisfy the consistency of coverage requirements of all targets. The authors proposed an algorithm used a pruning rule to avoid selecting redundant directions and selecting more than one direction of the same sensor for each cover set. In Reference 41, the authors explored the issue of target coverage where a subset of sensors should be allowed to track all targets while extending the lifetime of the network. Three centralized algorithms for sensors scheduling were proposed in which an LA would decide the sensors to be triggered to cover the available targets. In their proposed algorithm, the LA attempts to select the sensors that cover more targets and manage the critical targets to maximize the lifetime of the network. The list of available solutions and their proposed method are summarized in Table 1.

**TABLE 1** List of LA solutions and their proposed method

Article	Method
Javadi et al <sup>29</sup>	Topology control protocol based on LA (LBLATC). Uses distributed LA to reduced the energy consumption by dynamically optimizing the transmission range while keeping the connectivity
Mostafaei et al <sup>30</sup>	Distributed border surveillance algorithm. Distributed approach based LA to maximize the number of barriers and minimize the energy consumption for the border surveillance in WSNs
Mostafaei et al <sup>32</sup>	Partial coverage learning automata algorithm to implement sleep scheduling approaches to minimize the number of active sensors for covering the desired portion of the region of interest
Chand et al <sup>16</sup>	LA solution using the highest ratio of the remaining energy.
Mohamadi et al <sup>35</sup>	Designed a pruning rule to manage critical targets in order to maximize network lifetime, a pruning rule was specifically designed to avoid the selection of redundant sensors
Salleh and Marouf <sup>34</sup>	Disjoint and nondisjoint cover sets based LA algorithm for scheduling sensors
Mohamadi et al <sup>36</sup>	Irregular cellular learning automata algorithm to solving coverage problem in directional sensor networks
Mohamadi et al <sup>37</sup>	Proposed a heuristic called maximum network lifetime with adjustable ranges for directional sensors networks
Mohamadi et al <sup>38</sup>	LA-based algorithm that is able to generate cover sets containing sensors with appropriate sensing ranges
Mohamadi et al <sup>41</sup>	LA attempts to select the sensors that cover more targets and manage the critical targets to maximize the lifetime of the network

Since LA provides the sensor node with the purpose of learning their state and selecting their correct state, that is, either active or sleep mode, to extend the lifetime of the WSN's network. Therefore, this article leverages the simplicity of LA and proposes a novel adaptive pursuit learning based energy-efficient target coverage in WSNs. Although the theory of LA has been applied before to solve the problem of area coverage, our solution enjoys some desirable designed properties compared with Reference 14. In fact, we apply LA to each of the sensors to determine each state: active or sleep. Therefore, we opt to pursue the joint action of the team LA corresponding to the best solution found so far using the concept of pursuit learning.<sup>42-44</sup> Our proposed scheme is different from the work of Reference 14 as the LA update of latter work does not track the best solution found so far. In Reference 14, the authors implemented a reward and penalty mechanism in the following manner:

- Reward action sleep—if all targets of the sensor are covered. In other words, taking the opposite action, which is active, would not result in any benefit.
- Reward action activate—if the sensor in question covers at least one target exclusively. In other words, taking the opposite action, which is sleep, will result in at least one less target that cannot be covered.

In summary, the principal contributions of this article are as follows:

- We leverage the simplicity of LA and propose a novel adaptive pursuit learning based energy-efficient target coverage in WSNs.
- Our proposed solution enjoys some desirable designed properties compared with existing studies. In fact, we apply LA to each of the sensors to determine each state: active or sleep. Therefore, we opt to pursue the joint action of the team LA corresponding to the best solution found so far using the concept of pursuit learning.
- In order to fasten the speed of the algorithm and increase its scalability when the number of sensors is large, we propose a clustering algorithm that divides the main coverage problem into a set of smaller independent subproblems. In this manner, we can solve the subproblems in a parallel manner.

The remainder of this article is organized as follows. In Section 3, we introduce some background concepts about the theory of LA, which is an essential component in our solution. In Section 4, we explain our proposed novel adaptive pursuit learning based energy-efficient target coverage algorithm in WSN. Theoretical convergence results of the pursuit-LA and discretized learning automata (DLA) is also explained in Section 4. Further the clustering technique employed in this article is also explained in this section. Performance evaluation is carried out in Section 5. Finally, conclusions and future works are drawn in Section 6.

### 3 | BACKGROUND

In this section, we shall review some basic concepts about the theory of LA, which is fundamental for the understanding of our article.

#### 3.1 | Learning automata

In the field of automata theory, an automaton<sup>45-49</sup> is characterized as a quintuple made out of a set of states, a set of outputs or actions, an input, a function that maps the present state and the input to the following state, and a function that maps a present state (and input) into the current output. The following definitions are widely used in the field of LA.<sup>50,51</sup>

**Definition 1.** A LA is defined by a quintuple  $\langle A, B, Q, F(.,.), G(.) \rangle$ , where:

1.  $A = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$  is the set of outputs or actions that the LA must choose from.  $\alpha(t)$  is the action picked by the automaton at any moment or instant  $t$ .
2.  $B = \{\beta_1, \beta_2, \dots, \beta_m\}$  is the set of inputs or feedback to the automaton.  $\beta(t)$  is the feedback at any moment  $t$  corresponding to the chosen action. The set  $B$  can be limited or unbounded. The most widely recognized LA input information is  $B = \{0, 1\}$ , where  $\beta = 0$  represents reward or equivalently a positive feedback, and  $\beta = 1$  represents penalty or equivalently a negative feedback.
3.  $Q = \{q_1, q_2, \dots, q_s\}$  is the set of finite states, where  $Q(t)$  signifies the condition of the automaton at any moment  $t$ .
4.  $F(.,.): Q \times B \rightarrow Q$  is a mapping as far as the state and input at the moment  $t$ , with the end goal that,  $q(t+1) = F[q(t), \beta(t)]$ . It is known as a *transition function*, that is, a function that decides the condition of the automaton at any resulting time instant  $t+1$ . This mapping can either be deterministic or stochastic.
5.  $G(.)$ : is a mapping  $G: Q \rightarrow A$ , and is known as the *output function*.  $G$  decides the action taken by the automaton in the event that it is in a given state as:  $\alpha(t) = G[q(t)]$ .

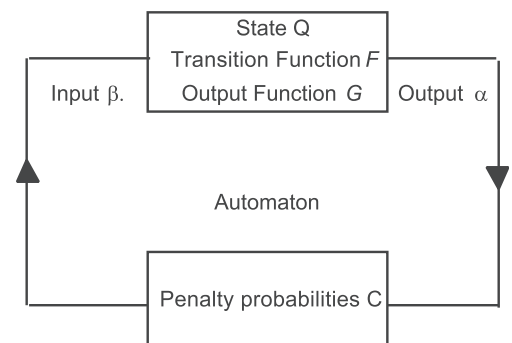
If the sets  $Q$ ,  $B$ , and  $A$  are all finite, the automaton is said to be *finite*.

The environment,  $E$ , ordinarily, alludes to the medium where the automaton functions. The Environment has all the outer variables that influence the activities of the automaton. Mathematically, an environment can be preoccupied with a triple  $\langle A, C, B \rangle$ .  $A$ ,  $C$ , and  $B$  are characterized as follows:

1.  $A = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$  is the set of actions.
2.  $B = \{\beta_1, \beta_2, \dots, \beta_m\}$  is the output set of the environment. Once more, we consider the situation when  $m=2$ , that is, with  $\beta = 0$  representing a "Reward," and  $\beta = 1$  representing a "Penalty."
3.  $C = \{c_1, c_2, \dots, c_r\}$  is a set of punishment or penalty probabilities, where component  $c_i \in C$  relates to an input activity  $\alpha_i$ .

The way toward learning depends on a learning loop, including the two entities: the random environment (RE), and the LA, as depicted in Figure 1. In the learning procedure, the LA persistently communicates with the environment to process reactions to its different activities (ie, its decisions). Finally, through adequate communications, the LA endeavors to gain proficiency with the ideal action offered by the RE. The real procedure of learning is represented as a set of associations or interactions between the RE and the LA.

The automaton is offered a set of actions, and it is obliged to picking one of them. At the point when an action is chosen among the pool of actions, the environment gives out a response  $\beta(t)$  at a time "t." The automaton is either penalized or rewarded with an obscure likelihood  $c_i$  or  $1 - c_i$ ,



**FIGURE 1** Feedback loop of LA. LA, learning automata

separately. Based on the response  $\beta(t)$ , the state of the automaton  $\varphi(t)$  is updated and another new action is picked at  $(t+1)$ . The penalty probability  $c_i$  satisfies:

$$c_i = \Pr[\beta(t) = 1 | \alpha(t) = \alpha_i] \quad (i = 1, 2, \dots, r). \quad (1)$$

We now present a few of the significant definitions utilized in the field.  $P(t)$  is alluded to as the action probability vector, where,  $P(t) = [p_1(t), p_2(t), \dots, p_r(t)]^T$ , in which every component of the vector:

$$p_i(t) = \Pr[\alpha(t) = \alpha_i], \quad i = 1, \dots, r, \text{ such that } \sum_{i=1}^r p_i(t) = 1 \quad \forall t. \quad (2)$$

Given an action probability vector,  $P(t)$  at time  $t$ , the average penalty is:

$$\begin{aligned} M(t) &= E[\beta(t)|P(t)] = \Pr[\beta(t) = 1|P(t)] \\ &= \sum_{i=1}^r \Pr[\beta(t) = 1 | \alpha(t) = \alpha_i] \Pr[\alpha(t) = \alpha_i] \\ &= \sum_{i=1}^r c_i p_i(t). \end{aligned} \quad (3)$$

The average penalty for the “pure-chance” automaton is given by:

$$M_0 = \frac{1}{r} \sum_{i=1}^r c_i. \quad (4)$$

As  $t \rightarrow \infty$ , if the average penalty  $M(t) < M_0$ , in any event asymptotically, the automaton is commonly viewed as superior to the pure-chance automaton.  $E[M(t)]$  is given by:

$$E[M(t)] = E\{E[\beta(t)|P(t)]\} = E[\beta(t)]. \quad (5)$$

A LA that performs better than by pure-chance is said to be expedient.

**Definition 2.** A LA is considered *expedient* if:

$$\lim_{t \rightarrow \infty} E[M(t)] < M_0.$$

**Definition 3.** A LA is said to be *absolutely expedient* if  $E[M(t+1) | P(t)] < M(t)$ , implying that  $E[M(t+1)] < E[M(t)]$ .

**Definition 4.** A LA is considered *optimal* if  $\lim_{t \rightarrow \infty} E[M(t)] = c_j$ , where  $c_j = \min_i \{c_i\}$ .

It ought to be noticed that no ideal LA exists. Marginally suboptimal performance, also termed above as  $\epsilon$ -optimal execution, is what that LA researchers endeavor to accomplish.

**Definition 5.** A LA is considered  $\epsilon$ -optimal if:

$$\lim_{n \rightarrow \infty} E[M(t)] < c_j + \epsilon, \quad (6)$$

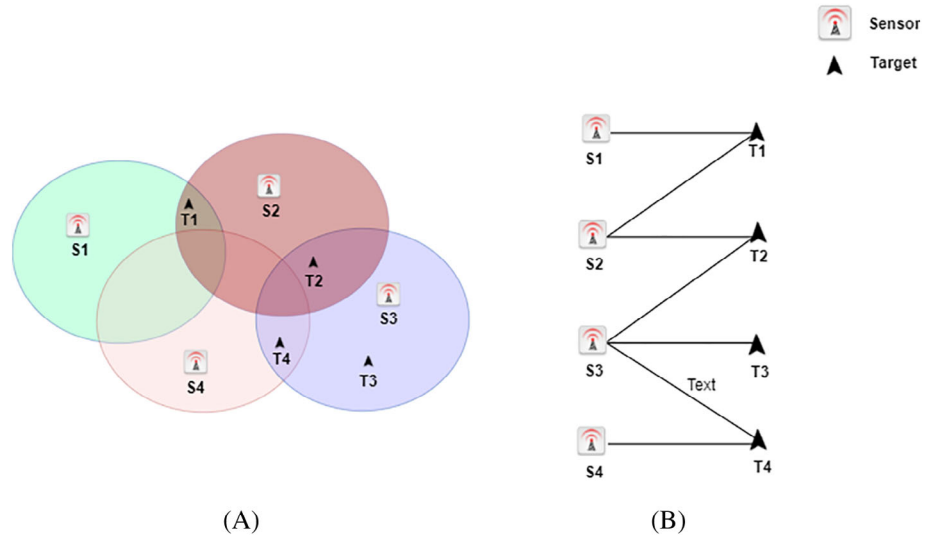
where  $\epsilon > 0$ , and can be arbitrarily small, by a reasonable choice of some parameter of the LA.

## 4 | PROPOSED NOVEL ADAPTIVE PURSUIT LEARNING ALGORITHM FOR TARGET COVERAGE IN WSN

### 4.1 | Problem formulation

Let us consider that there are a set of  $M$  targets denoted by  $T = \{T_1, T_2, \dots, T_M\}$  which are being monitored by a set of  $N$  sensor nodes  $S$  denoted by  $S = \{S_1, S_2, \dots, S_N\}$ . These two sets are deployed in a  $X \times X$  area. All sensor nodes have a fixed sensing range “R.” In addition, we assume that the

**FIGURE 2** A, Sensors coverage with their targets and B, bipartite graph of sensors and targets



number of sensor nodes exceeds the number of targets. In order to increase the lifetime of the network, a scheduling algorithm has been proposed in this article. A target point  $T_j$  within the range  $1 \leq j \leq M$ , is said to be covered by a sensor node if it falls inside this range of one of the sensor nodes  $1 \leq i \leq N$ .<sup>14</sup>

Figure 2 shows the Venn diagram of the sensors and the targets with their bipartite graph. There are four sensors S1, S2, S3, and S4 with their respective targets T1, T2, T3, and T4. The circle with different colors represents the coverage of each of the sensor nodes. The bipartite graph shows the relationship between the sensors and the number of covered targets by them. We can observe that by only activating two sensors: S2 and S3, we can cover all the targets. The complexity of the problem increases exponentially as we increase the number of sensors since the number of possible configurations of  $N$  sensors in active and sleep modes is  $2^N$ .

## 4.2 | Adaptive learning algorithm

We have used the LA algorithm for scheduling the sensor nodes. We shall now delineate the details of our algorithm, which helps in finding the best active set of sensors that are monitoring the maximum number of targets at any given instant. As in Reference 14, the actual flow of the algorithm is divided into three phases, which include an initial phase, a learning phase, and the target monitoring phase. We shall now focus on the initial phase and the learning phase. The monitoring phase is identical to the one presented in Reference 14, and therefore, it is omitted here for the sake of brevity.

### 4.2.1 | Initial phase

In this phase, each sensor node in the network is provided with LA, which helps the sensor node to select its state either to be active or sleep. In the initial stage, both states are equally probable, that is, the probability of selecting either of the active or sleep state is initialized to be 0.5. Here, sensor nodes are endowed with a certain level of autonomy permitting to establish communication, transfer messages, including their ID, position, and list of covered targets with their neighbor node autonomously. This phase is followed by the learning phase and the target monitoring phase.

### 4.2.2 | Learning phase

In the learning phase, each of the sensor nodes is equipped with LA. At first, the node is selected randomly. Using LA, each node selects its state. Then it broadcasts the message packet, including its all information to the rest of the sensor nodes.

**Algorithm 1.** Learning Phase

---

```

Best coverage set=  $\emptyset$ 
For each LA action set initial probability = 0.5
for Every sensor node in the network do
    Choose random action for sensor node
end for
while (iteration < max iterations) and (Convergence=False) do
    for Every sensor node in the network do
        Node= choose an action according to LA
        Update Current LA actions
        if Node state = Active then
            Current coverage set= Current coverage  $\cup$  node
        end if
    end for
    if (|Current coverage set| < |Best Coverage set|) and (All targets are covered) then
        Best coverage set=Current coverage set
        Best LA actions=Current LA actions
    end if
    for Every node in the network do
        if Best LA action of node is sleep then
            Decrease the probability to be active
        else
            Increase the probability of the node to be active
        end if
    end for
    iteration=iteration+1
    if All LA action probabilities are either  $\geq 1 - \epsilon$  or  $\leq \epsilon$  then
        Convergence=True
    end if
end while

```

---

We attach to each of the  $N$  sensors a LA. For instance, let us consider the sensor  $S_i$ . The automaton's state probability vector at the node  $i$  at time  $t$  is  $P_i(t) = [p_{(i,1)}(t), p_{(i,2)}(t)]$ .

Thus,  $p_{(i,j)}(t)$  is the probability at time instant  $t$  to select an action  $j$ . In our settings, we have two actions: sleep or active. For the sake of notation, let us denote 0 as the action sleep, and 1 denote the action active.

The feedback function is a binary function which yields a reward whenever the coverage has been improved. This is denoted by  $|Current coverage set| < |Best coverage set|$  in Algorithm 1. In more simple terms, if the aggregate state of the sensors chosen by the team of  $N$  LA yields an improvement in the coverage, which means ensuring full coverage with less number of sensors, the joint action of the LA that yielded that solution is rewarded by increasing the probability of the actions which formed that particular solution.

Let  $J = \{j_1(t), j_2(t), \dots, j_N(t)\}$  denote the action taken by the team of LA. Let  $J^* = \{j_1^*(t), j_2^*(t), \dots, j_N^*(t)\}$  be the best aggregate action of the team of LA so far yielding the highest coverage.

Thus, the idea of pursuit here is to reward the LA whose aggregate action is the highest possible so far, that is, till the time instant  $t$ .

### 4.2.3 | Update mechanism of the continuous learning automata

We consider the LA update equations at node  $i$ . The update is given by:

$$p_{(i,j)}(t+1) = (1-\lambda)\delta_{(i,j)} + \lambda p_{(i,j)}(t), \quad (7)$$



$$\delta_{(ij)} = \begin{cases} 1 & \text{if } j = j_i^*(t) \\ 0 & \text{else} \end{cases}, \quad (8)$$

$\lambda$  is the update parameter which is also called learning parameter and is time-independent. We shall also give some theoretical results for the case where  $\lambda$  is time-dependent. However, generally within the theory of LA, the most used schemes possess a learning parameter that is time-independent.

The informed reader would observe that the above update scheme corresponds to the linear reward-inaction LA update.<sup>27</sup> This pursuit paradigm is an adaptation of the PolyLA-Pursuit scheme<sup>52</sup> recently proposed by Goodwin and Yazidi in the context of machine learning classification problems. The pursuit LA used in this article is more suitable for solving deterministic optimization problem than classical LA used commonly in the literature to solve coverage problems that work better for stochastic optimization problems. Classical LA suppose that the environment response are stochastic, which translates in our context to a “stochastic” coverage results for a deterministic configuration. As this is not the case in our problem assumptions, meaning that the result of a configuration is deterministic in terms of coverage, it is more suitable to use our method pursuit LA. The pursuit-LA biases the search probability vector towards the optimal solution found so far. Furthermore, pursuit LA is faster and more accurate as it is by design suitable for solving deterministic optimization problems.

If  $j \neq j_i^*(t)$  then  $p_{(ij)}(t+1)$  is reduced by multiplying by  $\lambda$ , which is less than 1.

$$p_{(ij)}(t+1) = \lambda p_{(ij)}(t). \quad (9)$$

However, if  $j = j_i^*(t)$ , then  $p_{(ij)}(t+1)$  is increased by:

$$p_{(ij)}(t+1) - p_{(ij)}(t) = [(1 - \lambda) + \lambda p_{(ij)}(t)] - p_{(ij)}(t) \quad (10)$$

$$= (1 - \lambda) + p_{(ij)}(t)(\lambda - 1) \quad (11)$$

$$= (1 - \lambda)(1 - p_{(ij)}(t)) \geq 0. \quad (12)$$

In the rest of this article, we shall call the above update scheme as continuous learning automata (CLA) to differentiate it from the DLA which will be presented in Section 4.4.

The update scheme is called pursuit-LA<sup>52</sup> and has rules that obey the rules of the so-called linear reward-inaction LA. The idea is to always reward the transition probabilities along with the best solution obtained so far.

### 4.3 | Theoretical convergence results of the Pursuit-LA

We will now state some theoretical results that catalogue the properties of the Pursuit-LA for both the time varying update parameter and the fixed update parameter. The proofs of theorems can be found in the Appendix.

**Theorem 1.** *The optimal solution is generated with probability 1 only if the update parameter  $\lambda_t$  obeys the following condition:*

$$\sum_{t=1}^{\infty} \prod_{k=1}^{t-1} \lambda_k = \infty. \quad (13)$$

In Theorem 1, we give the convergence result of the Pursuit-LA for the case of fixed parameter  $\lambda$  that is time-dependent.

*Proof.* The proof of Theorem 1 is given in Appendix. ■

**Theorem 2.** *The optimal solution is generated with probability 1 only if the update parameter  $\lambda \rightarrow 1$ .*

*Proof.* The proof of Theorem 2 is given in Appendix. ■

In Theorem 2, we give the convergence result of the Pursuit-LA for fixed step-size parameter  $\lambda$ .

#### 4.4 | Introducing a new variant: DLA

Within the theory of LA, there are two main families of schemes, CLA and DLA.<sup>27</sup> In this regard, we present here a counterpart version of the CLA introduced in Section 4.2.

$$p_{(ij)}(t+1) = \Pi_H(p_{(ij)}(t) + \lambda\delta_{(ij)}), \quad (14)$$

where  $\delta_{(ij)}$  is given by Equation (8)

and where  $\Pi_H$  denote the following projection

$$\Pi_H(p) = \begin{cases} p, & \text{if } 1 < p < 0, \\ 1, & \text{if } p \geq 1, \\ 0, & \text{if } p \leq 0. \end{cases}$$

The discretized pursuit LA possesses the principles of the Pursuit-LA and is merely a discretized version of it that operates on a discrete probability space where the probability can take only a finite number of values, given that the increase and decrease has a fixed step. If  $j \neq j_i^*(t)$  then  $p_{(ij)}(t+1)$  is reduced by  $\lambda$ :

$$p_{(ij)}(t+1) = \text{Max}(p_{(ij)}(t) - \lambda, 0). \quad (15)$$

However, if  $j = j_i^*(t)$ , then  $p_{(ij)}(t+1)$  is increased by  $\lambda$ :

$$p_{(ij)}(t+1) = \text{Min}(p_{(ij)}(t) + \lambda, 1). \quad (16)$$

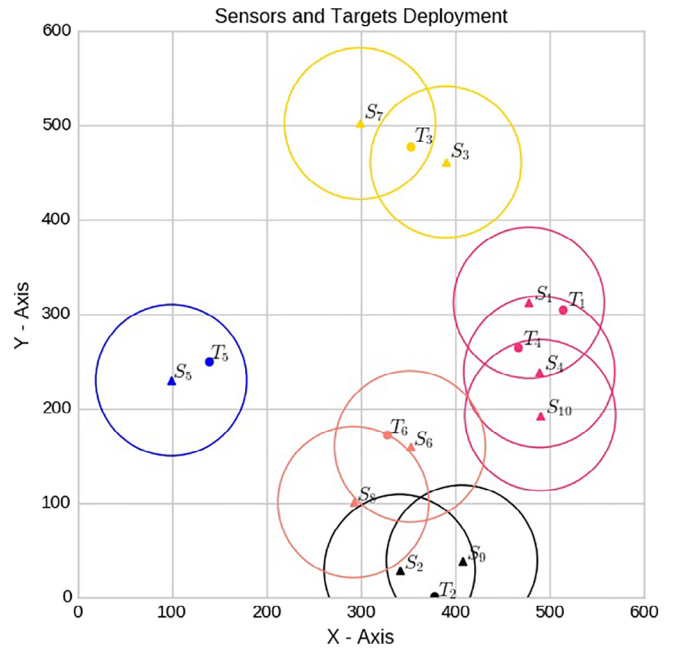
The informed reader would observe that in classical DLA scheme we rather use a resolution parameter that describes in loose terms the number of discrete values that the LA probability can take. In our case, it is easy to note that the number of discrete values is in the order of  $\lceil \frac{1}{\lambda} \rceil + 1$ .

The convergence proofs of the DLA are similar to those of the CLA found in the appendix and therefore are omitted for the sake of brevity.

#### 4.5 | Clustering for reducing problem complexity

The number of possible configurations of  $N$  sensors in active and sleep modes is  $2^N$  and this makes it practically impossible to find exact solutions for target coverage problems with more than few dozens of sensors as the computational time grows exponentially as a function of the number of sensors. However, in many cases it is possible to simplify the problem by splitting the sensors and their targets into smaller sets of independent clusters. For instance, if a target is covered by just a single sensor, these two objects may be separated from the rest and identified as a single cluster because the solution to the target coverage problem of this small cluster can be solved independently of the other targets and sensors. In this simplest possible case the only solution is that the sensor must be in active mode. In Figure 3, one such cluster can be seen colored in blue where the sensor  $S_5$  covers the target  $T_5$ . Another slightly more complex cluster is the yellow one where the sensors  $S_3$  and  $S_7$  cover target  $T_3$ . Again the solution of the coverage problem can be separated from the rest, only one of these two sensors needs to be active. When a single sensor covers two or more targets, all other sensors covering one or more of these targets must also be part of the same cluster, since a solution to the coverage problem of these targets must involve all these sensors. Such a case can be seen in Figure 3 where the sensors  $S_1$  and  $S_4$  cover the targets  $T_1$  and  $T_4$ . As sensor  $S_{10}$  also covers target  $T_4$ , this sensor must also be part of this pink cluster as its state is crucial in order to determine the optimal sensor configuration in order to cover the targets  $T_1$  and  $T_4$ . In this way, it is in general possible to split a complex configuration of sensors and targets into independent clusters and thereby reduce the coverage problem into subproblems which can be solved in parallel by virtue of the clustering mechanism. In the case presented in Figure 3, a target coverage problem with a solution space of  $2^{10}$  solutions has been reduced to 5 coverage problems with a solution space of 1, 2, 2, 2, and 8, respectively, for the five clusters. This enables us to find exact solutions for much larger problems than without clustering and obviously also improves the capabilities of the other algorithms when first reducing the complexity of the coverage problem at hand.

We present Algorithm 2 which finds all independent clusters for a given set of targets and sensors. Targets may in some cases be linked together in long chains leading to large clusters and it is essential that the algorithm loops through all possible targets which might be a candidate member for a cluster. As shown in the pseudo code of Algorithm 2, the while loop of the clustering algorithm keeps on including new targets until there are no more sensors which cover targets that have not yet been assigned members of the cluster under construction. Algorithm 3 is the pseudo code of

**FIGURE 3** Five clusters of sensors and targets

function `convert_into_cluster` which is called from Algorithm 2. From just a visual inspection of a graph like Figure 3, it is not obvious how to split the original set into clusters before the algorithm has performed the job and given each cluster an individual color.

---

**Algorithm 2.** Clustering of sensors and targets
 

---

Define global lists `visited_sensors` and `visited_targets = ∅`

Define global list `cluster = ∅`

**for** every target node **do**

  Define local lists `cluster_sensors` and `cluster_targets = ∅`

▷ to store targets and sensors of each cluster

**if** target not already visited **then**

`visited_targets = visited_targets ∪ target`

`cluster_targets = cluster_targets ∪ target`

`check_target_coverage = target`

**while** True **do**

      Define local list `found_new = ∅`

**for** every target in `check_target_coverage` **do**

        new sensors, new targets = call function `convert_into_cluster(sensors_list_covering_target)`

▷ Algorithm 3

`cluster_sensor = cluster_sensor ∪ new sensors`

`cluster_target = cluster_target ∪ new targets`

`found_new = found_new ∪ new targets`

**end for**

**if** size of `found_new` is greater than 0 **then**

`check_target_coverage = found_new`

**else**

        Break while loop

**end if**

**end while**

**end if**

**if** `cluster_targets` size is greater than 0 **then**

    Add `cluster_sensors`, `cluster_targets` to `cluster`

**end if**

**end for**

---

**Algorithm 3.** Function convert\_into\_cluster

---

```

Define function convert_into_cluster(list_of_sensors)
Define variable new sensors and new targets =  $\emptyset$ 
for each sensor in list of sensors that are covering current target do
  if sensor is not already visited then
    visited_sensors = visited_sensors  $\cup$  sensor
    new sensors = new sensors  $\cup$  sensor
    for every target covering new sensor list do
      if target is not already visited then
        visited_targets = visited_targets  $\cup$  target
        new targets = new targets  $\cup$  target
      end if
    end for
  end if
end for
end for
return the list of new sensors and targets associated with this cluster

```

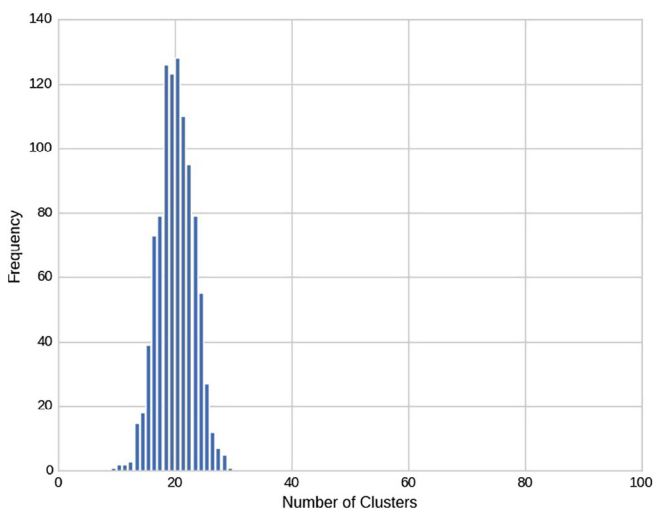
---

In general the solution space of the target coverage problems discussed in this article is too large to allow exact solutions. For small sets of sensors and targets, the proposed algorithms can be precisely evaluated by comparing to brute force (BF) algorithms computing the optimal exact solution. A major advantage of the clustering method is that it is possible to compute exact solutions to problems consisting of a large number of sensors and targets by generating subsets and finding exact solutions for each of them. Then other nonexact algorithms may be evaluated by addressing the initial large set problem and comparing their results to the optimal solution known because of the clustering. Without the clustering method these approximate algorithms could just be compared with each others, without knowing how far from the exact their results were.

#### 4.5.1 | Distribution of the cluster sizes

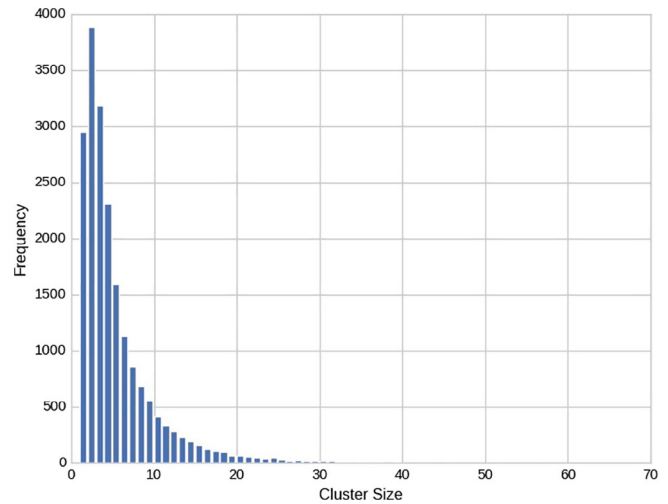
The clustering operation is a novel feature in the proposed solution that can speed up any sensor coverage algorithm. In this subsection, we would like to characterize the distribution of the cluster sizes, which gives us an idea about how much gain we could achieve by using clustering.

Figure 4 depicts the distribution of the number of clusters obtained for 1000 random configurations using a fixed number of targets and sensors, that is, 100 and 50, respectively, for a sensor range of 50. The sensors were deployed in a 600 by 600 grid. As seen in Figure 4, the most common case is that approximately 20 clusters are generated when applying the clustering algorithm. This means that the huge target covering problem of 50 targets and 100 sensors is typically reduced to 20 small problems containing just a handful of targets and sensors. Of course, if the sensor range is larger, the number of clusters will be smaller and the method will not be as efficient.



**FIGURE 4** The distribution of the number of clusters generated for 1000 random deployments of 100 sensors and 50 targets

**FIGURE 5** Cluster size distribution of all clusters created for 1000 random deployments of 100 sensors and 50 targets



**FIGURE 6** Dependence of cluster size on the sensor range. For each sensor range 1000 deployments of 100 sensors and 50 targets has been carried out. The error bars indicate the standard deviation of the distribution of cluster sizes

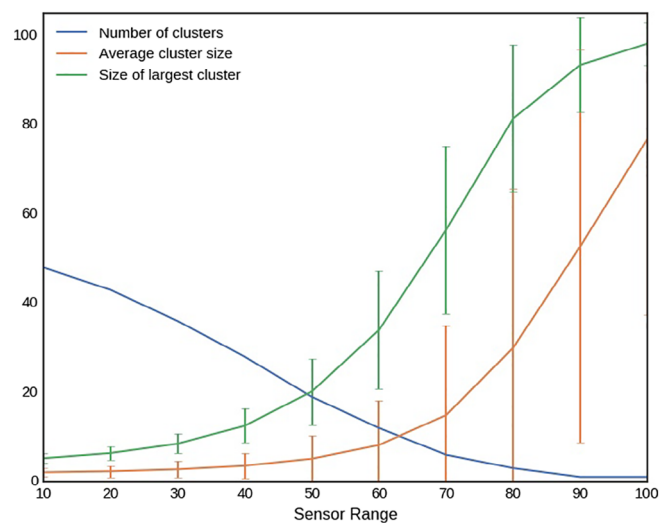
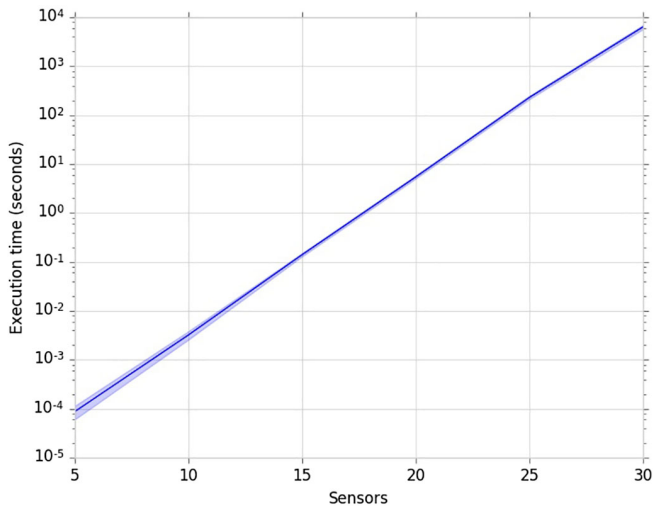


Figure 5 shows the distribution of cluster sizes for all the clusters generated in the same 1000 experiments of random deployment of 100 sensors and 50 targets. Most of the clusters are seen to be quite small; almost all of them are less than 20. The largest cluster contains 32 sensors and is a bit beyond the size, which can be solved using a BF algorithm. However, in most cases, the size is less than 20, making it possible to reduce the large coverage problem to many smaller problems that can be solved easily.

The size of the clusters generated when applying the clustering algorithm to deployment of sensors and targets, strongly depends on the sensor range. If the range is large, a single sensor will cover many targets, and this will lead to more entanglement and larger clusters. In Figure 6, the dependence on sensor range when generating the clusters is shown. For each of the sensor ranges shown in the figure, a thousand deployments of sensors were carried out, and clustering applied. As the sensor range is increased, the average number of clusters generated is reduced from almost 50 to close to 1. When the sensor range reaches 100, all sensors and targets are entangled, and this leads to a single cluster, meaning that it is not possible to reduce the complexity of the coverage problem. When solving a large coverage problem that has been reduced to many small problems in the form of clusters, the size of the largest of the clusters determines how hard it is to solve the entire problem. When the size of the largest cluster is 20 or less, the coverage problem can be solved using BF, computing all possible combinations of sensor activation. The green line shows how the size of the largest cluster depends on the sensor range. For a sensor range of 50, the average size of the largest cluster is 20. In the case of a cluster of 20 sensors, a BF solution is possible, and this will also be the case for smaller sensor ranges. As the sensor range increases, the error bars show that the standard deviation of the size of the largest cluster becomes quite large, and the same is the case for the average cluster size of the 1000 experiments. This makes sense as for small sensor ranges; clustering will systematically lead to many small clusters. For a very large sensor range, the result will mostly be just a single cluster and less variance. In between, the clustering will lead to both small and large clusters and a large variance in the cluster size.

Doing similar clustering experiments, one can establish which coverage problems can be addressed for a given number of sensors and targets and to what extent the coverage problems can be reduced to smaller and tractable problems.



**FIGURE 7** Average execution time of the brute force algorithm for 50 experiments

Abbreviation	Definition
BF	Brute force
C-BF	Cluster based brute force
CLA	Continuous learning automata
DLA	Discretized learning automata
C-CLA	Cluster based continuous learning automata

**TABLE 2** List of abbreviations used for algorithms

## 4.5.2 | BF algorithm

The BF algorithm is the process of finding the best solution by trying every possibility and selecting the one which gives the best result. It is a straightforward method to find the optimal result. For example, suppose we have four sensor nodes and five targets that need to be covered. Our goal is to find the minimum number of sensors that covers all five targets. In the case of the BF algorithm, the total number of possible solutions is given by  $2^n$  where  $n$  represents the input size. In our example, we have four sensors, which means  $n=4$ . This implies that we have  $2^4$ , that is, 16 solutions ranging from 0000, 0001, 0010 ... to 1111. Here, 0 represents that the sensor is in off state, and 1 represents that the sensor is active. By trying every solution, the BF algorithm finds the best solution that gives the minimum number of active sensors to cover all the targets.

From this example, we see that the total number of possible solutions is dependent on the input size. Since the number of possible solutions grows exponentially as a function of the input size, it cannot be used to solve a complex problem involving a large number of sensors as the number of alternatives gets too large for any computer to handle. To see the execution time of the BF algorithm, we have conducted 50 experiments for a number of sensors between 5 and 30, recorded the time to find the optimal solution, and the average results are presented in Figure 7. The shadowed area of the graph corresponds to a 95% confidence interval. Figure 7 shows that the BF execution time is very short for a small number of sensors. For instance, for five sensors, it takes only  $10^{-4}$  seconds to find the optimal result. For 18 sensors the algorithm spends 1 second, while for 30 sensors it takes slightly less than  $10^4$  seconds, which is almost  $10^8$  times larger than for five sensors. Most importantly, from Figure 7, we observe that the execution time of the BF algorithm has an exponential relationship with the number of sensors, as the logarithm of the execution time can be seen to be close to linear as a function of the number of sensors.

## 5 | EXPERIMENTS

In this section, we present our experimental results that demonstrate the effectiveness of our approach. The simulations were performed using a customized simulation environment implemented in Python. For the sake of clarity, we summarize all the abbreviations in Table 2. In all the experiments, we use a grid size of  $600 \times 600$ .

## 5.1 | Accuracy of the algorithm

In this section, we compare the accuracy of the cluster-based *C*-CLA algorithm with its counterpart solution CLA (which does not involve clustering) and with the optimal solution obtained with the brute force algorithm (*C*-BF). We report two major experiments, a small scale experiment with a maximum of 100 sensors and a large-scale experiment with a maximum of 300 sensors.

### 5.1.1 | Small scale experiments

In these experiments, the learning parameters are set to  $\lambda = 0.9999$  and  $\epsilon = 0.01$  while the sensor range is set to 50. We vary the number of sensors by a step size of 10 from 10 sensors to 100 sensors and choose the number of targets to be half the number of sensors. To obtain reliable results, we test 30 different configurations at random for each step size of sensor pools.

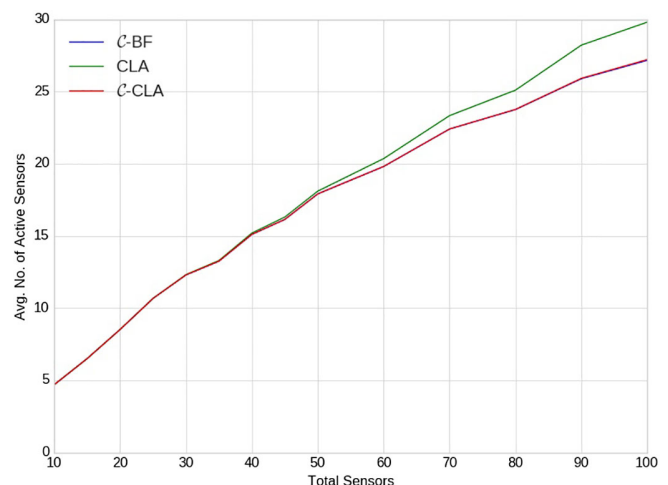
Our proposed *C*-CLA algorithm yields an optimal number of active sensors even as the size of the sensor pool increases to 100, as seen in Figure 8. The results of the *C*-CLA algorithm are shown as a red line and are for all experiments the results equal the exact optimal results of the *C*-BF algorithm. Therefore, blue *C*-BF line is covered by the red *C*-CLA line. The performance of CLA, however, decreases as we increase the number of sensors. In this context, the CLA's accuracy declines by producing more active sensors than the optimal number of sensors depicted by the red line starting from 40 sensors. The error steadily increases beyond 40 sensors. While Figure 8 shows the performance of our algorithms using the average number of active sensors as a metric, Figure 9 emphasizes the accuracy of the algorithms compared with the exact results of the *C*-BF algorithm. A 100% accuracy describes the case where the number of active sensors coincides with the optimal solution. The accuracy is calculated as

$$\text{Accuracy} = 1 - \text{Error} = 1 - \frac{\text{Number of active sensors} - \text{Optimal number of active sensors}}{\text{Optimal number of active sensors}}. \quad (17)$$

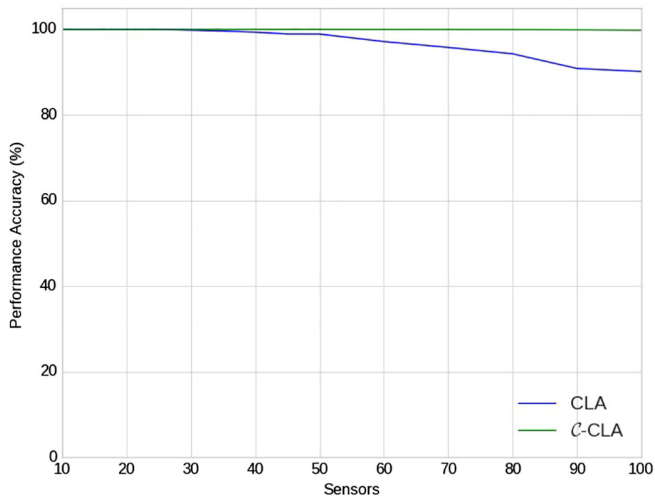
The difference of accuracy between *C*-CLA and CLA is seen in Figure 9 to increase from zero for 30 sensors to a maximum of 10% at 100 sensors. For this number of sensors, the accuracy of the CLA algorithm is around 90% while the accuracy of *C*-CLA is a 100%, the performance gap being 10%.

### 5.1.2 | Large-scale experiments

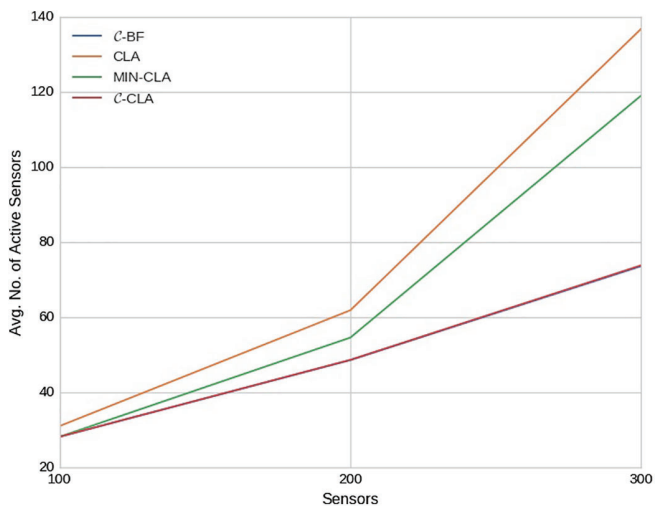
Although the *C*-CLA algorithm achieves optimal performance in the previous experiment when the maximum number of sensors is 100, we cannot claim that it is an optimal heuristic. In this experiment, by resorting to a number of sensors beyond 100 sensors, we increase the scale of the experiment and measure its performance. Using 100, 200, and 300 sensors with respective ranges 50, 40, and 30, we compare the *C*-CLA, CLA, and *C*-BF algorithms as shown in Figure 10. For these relatively small sensor ranges it is still possible to find exact solutions using the *C*-BF algorithm. For each setup, the experiments were done 100 times. To gain a better insight into the CLA's performance, we report both the CLA's average performance as well as the best-achieved result characterized by MIN-CLA, that is, the minimum number of sensors among the 100 runs obtained. Interestingly, the *C*-CLA algorithm remains ideal and have an accuracy that is identical to that of the *C*-BF. The performance of the *C*-BF algorithm corresponding to a blue line is identical to the performance of *C*-CLA presented by a purple color line and hence covered by the latter line. CLA performance tends to decline, however, and even MIN-CLA's best-case scenario for CLA is far from optimal. For instance, when the number of sensors is 300, the CLA



**FIGURE 8** Average number of active sensors. The red *C*-CLA line covers the blue *C*-BF line as the results are identical. *C*-BF, cluster based brute force; *C*-CLA, cluster based continuous learning automata



**FIGURE 9** Performance accuracy of the  $C$ -CLA and CLA algorithms.  $C$ -CLA, cluster based continuous learning automata; CLA, continuous learning automata



**FIGURE 10** Average number of active sensors in large-scale systems. The purple  $C$ -CLA line covers the blue  $C$ -BF line as the results are identical.  $C$ -BF, cluster based brute force;  $C$ -CLA, cluster based continuous learning automata

provides a number of active sensors, which is twice the optimal number, that is, around 136, while the optimal result is 73. Even CLA's best-case scenario, MIN-CLA, gives around 119 active sensors and is far from the exact result.

## 5.2 | Effect of the $\lambda$ parameter on the performance in terms of accuracy and execution time

In these experiments, we test the effect of the learning parameter  $\lambda$  on the convergence time and accuracy of our LA algorithms. We use different values of  $\lambda$  in this experiment ranging from  $\lambda = 0.9$ ,  $\lambda = 0.99$ ,  $\lambda = 0.999$ , to  $\lambda = 0.9999$ . The sensor pool size is varied by a step size of 10 from 10 to 100 sensors. The range of the sensors is kept constant at 50, and the target numbers are chosen to be half the number of sensors. For each configuration, 100 experiments are performed. Figure 11 shows the average number of active sensors for different values of  $\lambda$  for the  $C$ -BF, CLA, and  $C$ -CLA algorithms. Please note that the average number of active sensors obtained from 1000 experiments is each number in the graphs<sup>1</sup>.

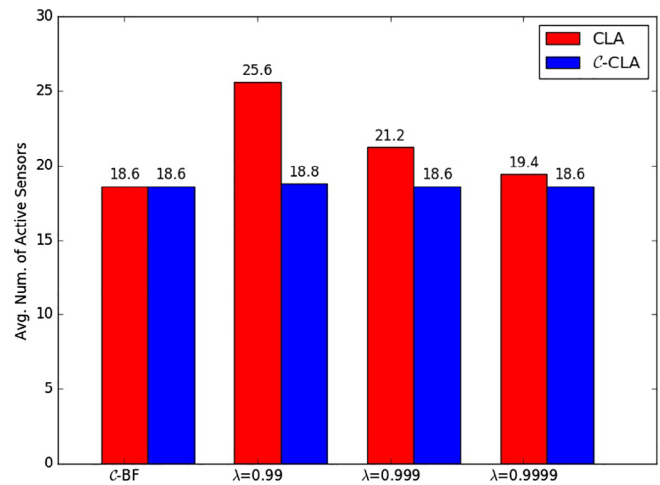
As seen in Figure 11, the  $C$ -BF algorithm offers an average number of active sensors as the optimal solution for all targets, whereas for  $\lambda = 0.99$ , our proposed  $C$ -CLA provides 18.8 as an average number of active sensors which is only 1% higher than the optimal solution, whereas CLA achieves 25.6 which is about 37.6% higher than the optimal solution. However, for both  $\lambda = 0.999$  and  $\lambda = 0.9999$ ,  $C$ -CLA offers an average of 18.6 sensors, which is the optimal performance, while CLA gives 21.2 and 19.4, respectively. Based on this, we can conclude that for the  $C$ -CLA algorithm,  $\lambda = 0.999$  is a small enough value to achieve 100% accuracy.

Figure 12 provides a closer view of the effect of varying the  $\lambda$  value on CLA and  $C$ -CLA performance for different sensor pool sizes. The CLA algorithm efficiency is highly sensitive to changes in the value of the parameter  $\lambda$ , as can be seen in Figure 12. All of the three  $\lambda$  values provide an

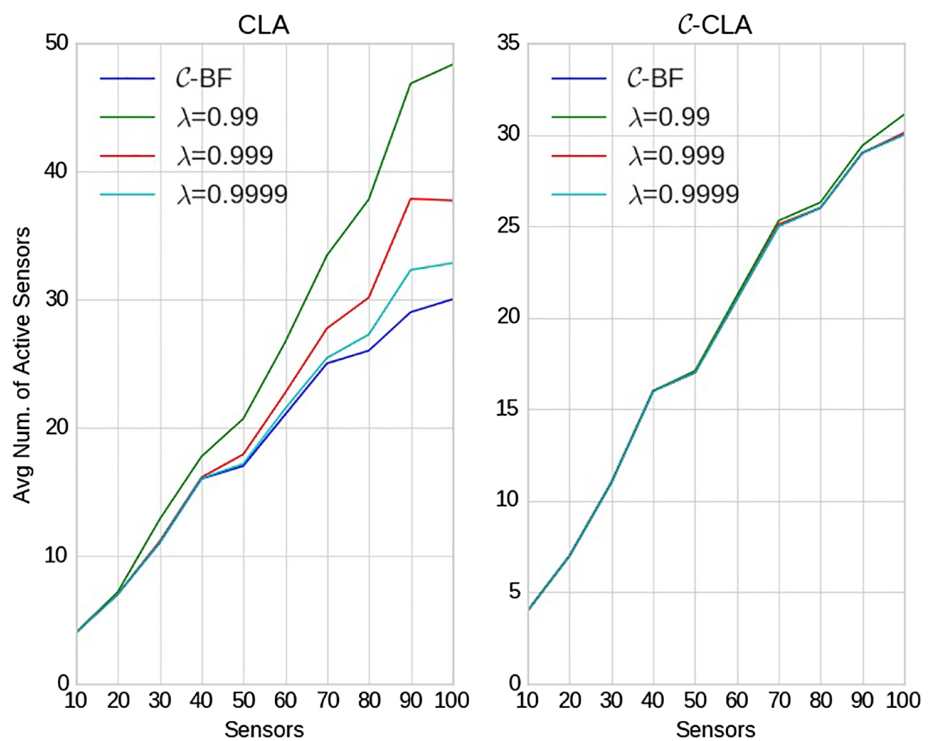
<sup>1</sup>The number of experiments corresponds to 1000 since we use 10 different sizes of sensor pools and 100 experiments for every single size



**FIGURE 11** The performance of the CLA and  $\mathcal{C}$ -CLA algorithms as function of  $\lambda$ .  $\mathcal{C}$ -CLA, cluster based continuous learning automata; CLA, continuous learning automata

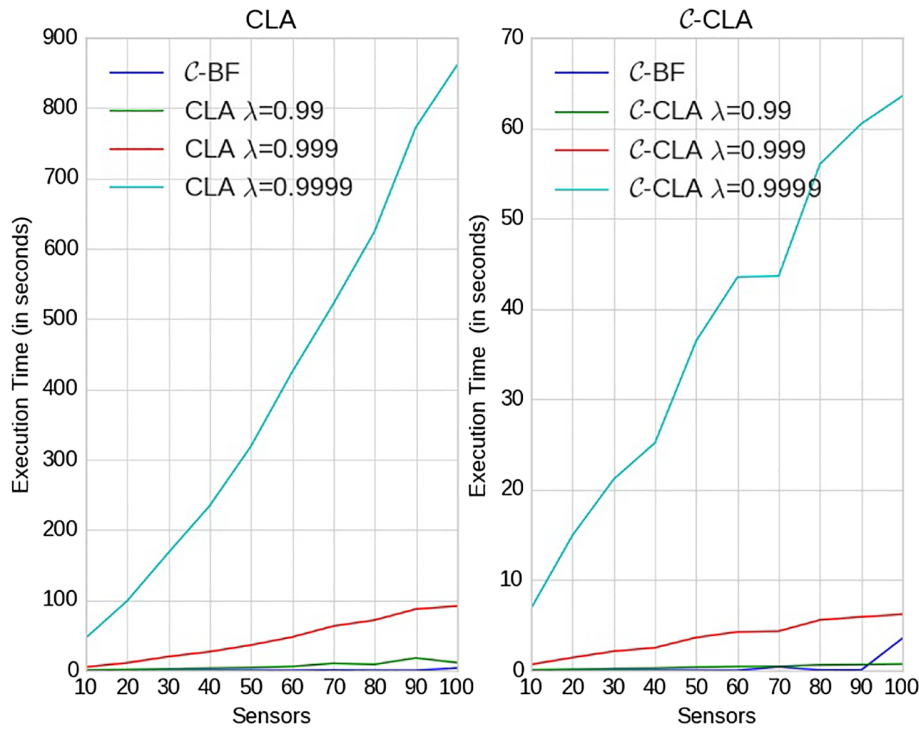


**FIGURE 12** Effect of the  $\lambda$  value on CLA and  $\mathcal{C}$ -CLA.  $\mathcal{C}$ -CLA, cluster based continuous learning automata; CLA, continuous learning automata



optimal solution for sensor pool sizes of 10 and 20. When starting from a sensor pool size larger than 20 sensors, the performance of  $\lambda = 0.99$  is increasingly inferior compared with the exact  $\mathcal{C}$ -BF results. Likewise, in the case of CLA,  $\lambda = 0.999$  and  $\lambda = 0.9999$  are also far from the optimal results for more than 40 sensors. On the other hand, the proposed  $\mathcal{C}$ -CLA algorithm keeps up with the optimal solution obtained by  $\mathcal{C}$ -BF for all three values of  $\lambda$  up to around 70 sensor nodes. Beyond 70 sensors, the  $\mathcal{C}$ -CLA performance begins to decline for only  $\lambda = 0.99$ , with results slightly deviating from the optimal ones. The other  $\lambda$  values,  $\lambda = 0.999$  and  $\lambda = 0.9999$ , continue to deliver optimal results. We further compare the efficiency of both algorithms, measured in seconds in terms of execution time. Figure 13 displays the execution time taken by the algorithms CLA and  $\mathcal{C}$ -CLA for the specific values of  $\lambda$ .

According to Figure 13, the execution time spent by the CLA algorithm is higher than the proposed  $\mathcal{C}$ -CLA algorithm for different values of  $\lambda$  by order of magnitude. CLA takes about 860 seconds for 100 sensors and  $\lambda = 0.9999$  while  $\mathcal{C}$ -CLA takes about 64 seconds. But if we compare the average time taken for the first time to find the optimal solution, the proposed  $\mathcal{C}$ -CLA takes only about 6 seconds, the rest of the time is spent on reaching the termination condition. Since we use a small learning parameter  $\lambda = 0.999$ , it is easy to conclude that the execution time of the  $\mathcal{C}$ -CLA will be considerably lower, since all components of the probability vector need to reach  $1 - \epsilon$  or  $\epsilon$  starting from the initial probability 0.5.



**FIGURE 13** Execution time of CLA and *C*-CLA. *C*-CLA, cluster based continuous learning automata; CLA, continuous learning automata

$\lambda = 0.99$		$\lambda = 0.999$		$\lambda = 0.9999$	
CLA-I	<i>C</i> -CLA-I	CLA-I	<i>C</i> -CLA-I	CLA-I	<i>C</i> -CLA-I
450.06	271.06	4131.2	2617.66	39 208.4	26 087.06
635.38	314.56	4739.6	3130.90	41 592.4	31 298.02
762.21	323.85	5781.0	3204.96	50 639.4	32 012.05
856.99	339.00	6877.6	3358.17	59 500.6	33 537.82
930.02	345.13	7529.8	3368.54	66 752.4	33 545.47
1111.54	357.11	7918.2	3486.75	72 719.4	34 779.911
1290.10	365.95	9241.0	3497.26	75 564.4	34 825.77
1461.53	369.62	9112.2	3548.75	83 617.0	35 047.63
2093.42	343.13	10 676.8	3318.76	85 768.0	32 602.57
2476.00	368.57	11 126.4	3453.47	85 699.8	33 201.11

**TABLE 3** Number of Iteration taken by CLA and *C*-CLA over  $\lambda = 0.99$ ,  $\lambda = 0.999$ , and  $\lambda = 0.9999$

Abbreviation: *C*-CLA, cluster based continuous learning automata.

Let  $t_0$  be the minimum number of iterations to reach  $\epsilon$  from 0.5; then it is easy to note from the proof of Theorem 2 that  $t_0$  can be solved by solving the following equation.

$$0.5\lambda^{t_0} = \epsilon.$$

Similarly, Let  $t_1$  be the minimum number of iteration to reach  $1 - \epsilon$  from 0.5, which can be found by solving the following equation.

$$0.5 + \frac{1 - \lambda^{t_1}}{1 - \lambda} = 1\epsilon$$

Table 3 shows how many iterations the proposed *C*-CLA and CLA algorithms require to converge the probability to 1 or 0. We note the *C*-CLA takes less iteration than CLA. The CLA takes 2476, 11 126.4, and 85 699.8 iterations over  $\lambda = 0.99$ ,  $\lambda = 0.999$ , and  $\lambda = 0.999$ , while *C*-CLA only takes 368.57, 3453.47, and 33 201.11, respectively.

### 5.3 | Network life-time

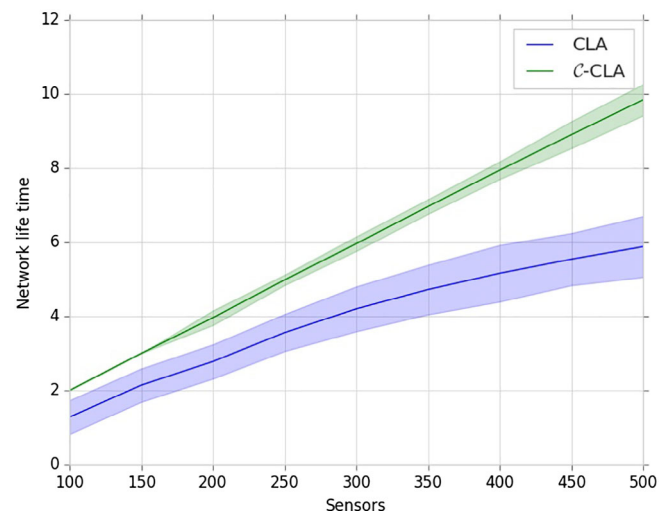
Network life-time is always considered as one of the most important parameters in WSN applications because of use of small batteries. Recharging or replacing batteries is not always feasible in a hostile environment or network with a large number of nodes, and so on.<sup>11,53</sup> We define each sensor to have a battery capacity of 1 time unit and the network life-time to be the number of time units over which the sensor network can cover *all* of its targets.<sup>54</sup> Prolonging the network life-time in this case reduces to maximizing the number of disjunctive groups of sensors, meaning with no common element, that each covers all the targets.<sup>14</sup> In this article, we adopt a “peeling-off” procedure to find those disjunctive groups and consequently compute the life-time. In simple terms, we use our pursuit-LA solution to find a group of sensors to be active for 1 time unit, then those sensors are removed and the network life-time is increased by one. Afterward, the procedure continues by “peeling-off” the group for which the battery was exhausted, and proceeding further to the next group. The procedure continues until we reach a point that we cannot ensure full coverage with the remaining sensors that are left. Therefore, to check the efficiency of our proposed *C*-CLA algorithm in terms of network life-time, we conducted an experiment where we varied the number of sensors starting from 100 with step size 50 up to 500. In each case 50 experiments were performed and the sensor range and the number of target were set to 50 as fixed parameters throughout the experiments.

Figure 14 clearly shows that when the number of sensors is 100, the difference between the average network life-time of the two algorithms is quite small, but as the number of sensors increases, the gap between the two network life-times also becomes larger. For 500 sensors, the network life-time computed by our proposed *C*-CLA algorithm is almost 10, whereas for the CLA algorithm the result is nearly 6. This means that our proposed *C*-CLA algorithm on average adds four more units of a life-time to the network. Another difference to observe from the experiments is the variation of the computed life-times. The shadowed area in Figure 14 represents a 95% confidence interval and the variation of the results of our proposed *C*-CLA algorithm is smaller than for the results of the CLA algorithm.

### 5.4 | Comparison of CLA and DLA approach

In this section, we compare both *C*-CLA and its counter-part *C*-DLA where *C*-DLA stands for Cluster-based DLA. In order to allow a fair comparison of both algorithms, we follow the same experimental approach for comparing learning algorithms as in Reference 55 and determine the respective tuning parameter for *C*-CLA and *C*-DLA that yields 99% of the time an optimal solution over a set of 100 experiments. Then we report the number of iterations to achieve this result. As we see in Section 5.2, the value of  $\lambda$  plays an important role in finding optimal results, so we have created an automated system that increases or decreases the value of  $\lambda$  to find out the accuracy of 99% using experimental procedure.<sup>55</sup> Table 4 represents the results of the CLA and DLA experiments.

From Table 4, we note that CLA and DLA do not have a clear winner. DLA takes fewer iterations in the first two experiments to converge to 99% solution precision while CLA is the winner in the next three tests. After seeing results, we find out that the number of CLA and DLA iterations depends not only on the value of  $\lambda$  but also on random sensor selection.

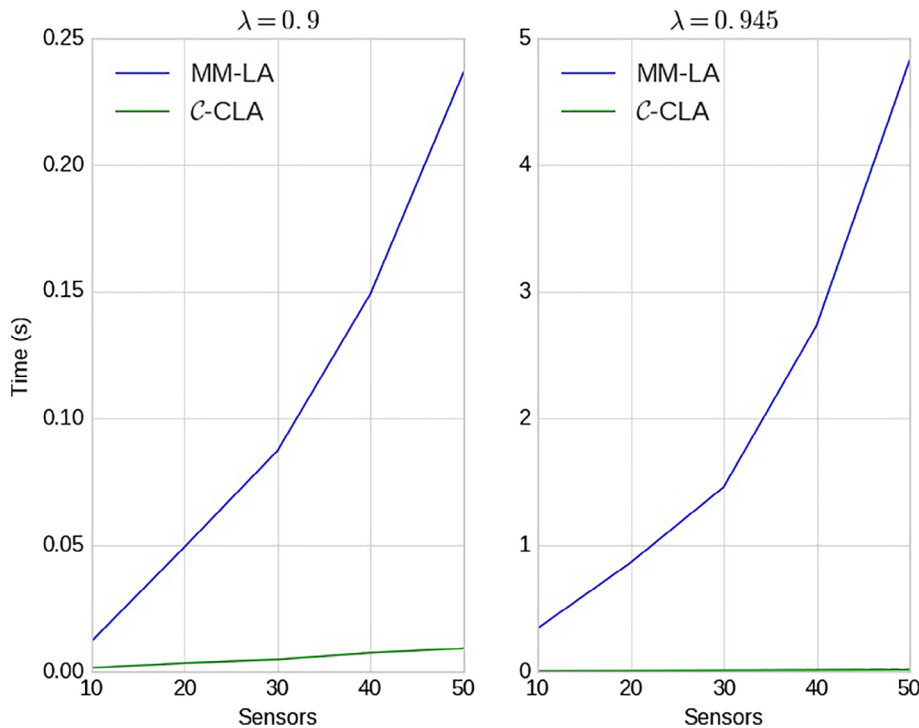


**FIGURE 14** Network lifetime of CLA and *C*-CLA. *C*-CLA, cluster based continuous learning automata; CLA, continuous learning automata

CLA Iterations	$\lambda$ Value	DLA Iterations	$\lambda$ Value
8599	0.00055	7963	0.0001
9347	0.00055	9033	0.0001
618	0.01	1812	0.00055
421	0.015625	1064	0.001
5101	0.001	8656	0.0001
9190	0.00055	8848	0.0001

**TABLE 4** Results obtained for 20 sensors and 10 targets with fix sensor sensing range 50 m

Abbreviations: CLA, continuous learning automata; DLA, discretized learning automata.



**FIGURE 15** Execution time of MM-LA and C-CLA at  $\lambda = 0.9$  and  $\lambda = 0.945$ . C-CLA, cluster based continuous learning automata; LA, learning automata

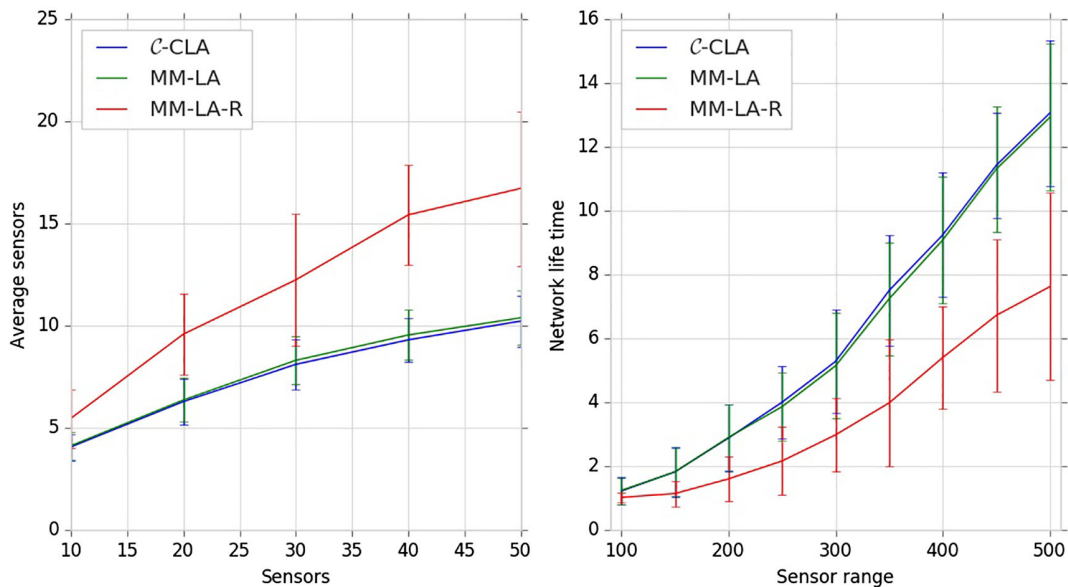
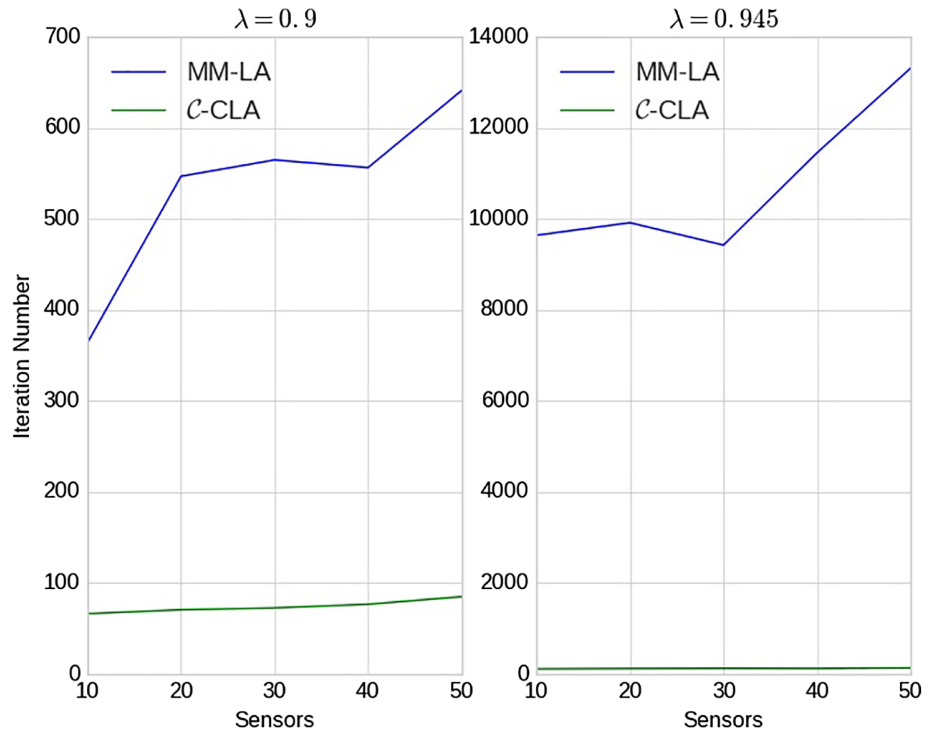
## 5.5 | Comparison of the proposed C-CLA algorithm with MM-LA

In this section, we compare our work to the approach presented by Mostafaei and Meybodi.<sup>14</sup> We call the latter algorithm MM-LA, where the acronym MM stands for Mostafaei and Meybodi. We resort to a number of sensors ranging from 10 to 50 (with steps of 10) for this experiment, and we let the number of targets be half the corresponding sensor pool size. Ten different random deployment of sensors and targets are used to obtain statistically reliable results. See Figures 15 and 16 for the average results.

According to Figure 15, the execution time of the proposed C-CLA algorithm is always lower than MM-LA for  $\lambda = 0.9$  and  $\lambda = 0.945$  using different pool sizes of sensors. Likewise, Figure 16 illustrates the number of iterations taken to converge the probability of sensors to 1 or 0. Interestingly our proposed algorithms are also much better than MM-LA in this experiment. We note that MM-LA requires about 640 and 13 300 iterations for  $\lambda = 0.9$  and  $\lambda = 0.945$ , respectively, for a sensor pool size of 50, whereas C-CLA only needs about 85 and 135 iterations per cluster, respectively.

Furthermore, in Figure 17, the left figure shows the comparison in terms of finding the average number of active sensors, and the right figure shows the comparison of the network life-time. In both cases, we can observe that the C-CLA algorithm performs better. It should be noted that for the MM algorithm, there are redundant sensors covering the already covered targets. Here, MM-LA-R represents the MM algorithm result with redundancy, that is, redundant sensors covering the same covered targets. The MM algorithm uses redundancy checking and removes this redundancy in the final result. Interestingly, after removing redundancy (as indicated by MM-LA in Figure 17) our proposed C-CLA algorithm still has slightly better performance.

**FIGURE 16** Number of Iterations of MM-LA and *C*-CLA for  $\lambda = 0.9$  and  $\lambda = 0.945$ . *C*-CLA, cluster based continuous learning automata; LA, learning automata



**FIGURE 17** Comparison of the two algorithms in terms of average active sensors and network life time

## 6 | CONCLUSION AND FUTURE WORKS

In this article, we focused on solving the problem of target coverage in WSNs in a better way and proposed the *C*-CLA algorithm, a novel-solution based on a cluster of sensors and targets using LA to help the sensor determine whether to be active or to sleep autonomously using the LA pursuit concept. Not only does our proposed algorithm find the optimal solution for covering all targets, it also takes minimal execution time compared with existing solutions.

The proposed *C*-CLA algorithm can provide 100% accuracy, based on the result presented in Section 5. Similarly, in different situations, the proposed *C*-CLA algorithm is very fast in terms of execution time including small to high sensor pool size and different lambda values. Interestingly, *C*-CLA needs a fewer number of iterations and spends less time to determine the optimal results compared with other existing<sup>14</sup> approaches.

Through our experimental testing, it is evident that our proposed adaptive learning algorithm is a feasible and viable approach for the sensor nodes to select their appropriate “sleep” and “active” state autonomously for energy-efficient target coverage in the WSNs.

For future work, it would be interesting to apply our approach for the sensors connectivity problem with mobile targets. In addition, the application of game theory to the existing approach would be an interesting research direction.

## ACKNOWLEDGMENT

A preliminary version of this article was presented in Reference 56.

## ORCID

Anis Yazidi  <https://orcid.org/0000-0001-7591-1659>

## REFERENCES

- Whitmore A, Agarwal A, Da Xu L. The Internet of Things—a survey of topics and trends. *Inf Syst Front*. 2015;17(2):261-274.
- Alaba FA, Othman M, Hashem IAT, Alotaibi F. Internet of Things security: a survey. *J Netw Comput Appl*. 2017;88:10-28.
- Rauniyar A, Engelstad P, Moen J. A new distributed localization algorithm using social learning based particle swarm optimization for Internet of Things. Paper presented at: Proceedings of the 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), Porto, Portugal: IEEE; 2018:1-7.
- Weir GE, Center UNH. The American sound surveillance system: using the ocean to hunt Soviet submarines, 1950–1961. *Int J Naval History*. 2006;5(2):1–20.
- Demigha O, Hidouci WK, Ahmed T. On energy efficiency in collaborative target tracking in wireless sensor network: a review. *IEEE Commun Surv Tutor*. 2012;15(3):1210-1222.
- Luo Y, Duan Y, Li W, Pace P, Fortino G. Workshop networks integration using mobile intelligence in smart factories. *IEEE Commun Mag*. 2018;56(2):68-75.
- Onur E, Ersoy C, Deliç H, Akarun L. Surveillance wireless sensor networks: deployment quality analysis. *IEEE Netw*. 2007;21(6):48-53.
- Mini S, Udgata SK, Sabat SL. Sensor deployment and scheduling for target coverage problem in wireless sensor networks. *IEEE Sens J*. 2013;14(3):636-644.
- Gungor VC, Hancke GP. Industrial wireless sensor networks: challenges, design principles, and technical approaches. *IEEE Trans Ind Electron*. 2009;56(10):4258-4265.
- Othman MF, Shazali K. Wireless sensor network applications: a study in environment monitoring system. *Proc Eng*. 2012;41:1204-1210.
- Rauniyar A, Engelstad PE, Østerbø ON. Performance analysis of RF energy harvesting and information transmission based on NOMA with interfering signal for IoT relay systems. *IEEE Sens J*. 2019;19(17):7668-7682. <https://doi.org/10.1109/JSEN.2019.2914796>.
- Thomas D, Shankaran R, Orgun M, Hitchens M, Ni W. Energy-efficient military surveillance: coverage meets connectivity. *IEEE Sens J*. 2019;19(10):3902-3911.
- Nguyen NT, Liu BH. The mobile sensor deployment problem and the target coverage problem in mobile wireless sensor networks are NP-hard. *IEEE Syst J*. 2018;13(2):1312-1315.
- Mostafaei H, Meybodi MR. Maximizing lifetime of target coverage in wireless sensor networks using learning automata. *Wirel Personal Commun*. 2013;71(2):1461-1477.
- More A, Raisinghani V. A survey on energy efficient coverage protocols in wireless sensor networks. *J King Saud Univ-Comput Inf Sci*. 2017;29(4):428-448.
- Chand S, Kumar B. Target coverage heuristic based on learning automata in wireless sensor networks. *IET Wirel Sens Syst*. 2018;8(3):109-115.
- Elhabyan R, Shi W, St-Hilaire M. Coverage protocols for wireless sensor networks: review and future directions. *J Commun Netw*. 2019;21(1):45-60.
- Zou T, Li Z, Li S, Lin S. Adaptive energy-efficient target detection based on mobile wireless sensor networks. *Sensors*. 2017;17(5):1028.
- Gupta SK, Kuila P, Jana PK. Genetic algorithm approach for k-coverage and m-connected node placement in target based wireless sensor networks. *Comput Electr Eng*. 2016;56:544-556.
- Pan JS, Lin JCW, Sui B, Tseng SP. Genetic and evolutionary computing. Paper presented at: Proceedings of the Twelfth International Conference on Genetic and Evolutionary Computing, December 14-17, 2019:834; Changzhou, Jiangsu, China, Springer.
- More A, Raisinghani V. A node failure and battery-aware coverage protocol for wireless sensor networks. *Comput Electr Eng*. 2017;64:200-219.
- Aranzazu-Suescun C, Cardei M. Energy-efficient weak-barrier coverage with adaptive sensor rotation. *J Combinat Optim*. 2019;1-19.
- Elhoseny M, Tharwat A, Yuan X, Hassanién AE. Optimizing K-coverage of mobile WSNs. *Exp Syst Appl*. 2018;92:142-153.
- Lersteau C, Rossi A, Sevaux M. Minimum energy target tracking with coverage guarantee in wireless sensor networks. *Europ J Operat Res*. 2018;265(3):882-894.
- Tripathi A, Gupta HP, Dutta T, Kumar D, Jit S, Shukla K. A target tracking system using directional nodes in wireless sensor networks. *IEEE Syst J*. 2018;13(2):1618-1627.
- Chaturvedi P, Daniel A. A novel approach for target coverage in wireless sensor networks based on network coding. *Soft Computing: Theories and Applications*. Singapore: Springer; 2019:303-310.
- Narendra KS, Thathachar MA. Learning automata—a survey. *IEEE Trans Syst Man Cybern*. 1974;SMC-4(4):323-334.
- Mostafaei H, Chowdhury MU, Islam R, Gholizadeh H. Connected P-Percent coverage in wireless sensor networks based on degree constraint dominating set approach. Paper presented at: Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems. Cancun, Mexico: ACM; 2015:157-160.
- Javadi M, Mostafaei H, Chowdhury MU, Abawajy JH. Learning automaton based topology control protocol for extending wireless sensor networks lifetime. *J Netw Comput Appl*. 2018;122:128-136.
- Mostafaei H, Chowdhury MU, Obaidat MS. Border surveillance with WSN systems in a distributed manner. *IEEE Syst J*. 2018;12(4):3703-3712.
- Mostafaei H, Shojafar M, Zaher B, Singhal M. Barrier coverage of WSNs with the imperialist competitive algorithm. *J Supercomput*. 2017;73(11):4957-4980.
- Mostafaei H, Montieri A, Persico V, Pescapé A. A sleep scheduling approach based on learning automata for WSN partial coverage. *J Netw Comput Appl*. 2017;80:67-78.

33. Mostafaei H, Obaidat MS. A greedy overlap-based algorithm for partial coverage of heterogeneous wsns. Paper presented at: Proceedings of the GLOBECOM 2017-2017 IEEE Global Communications Conference. Singapore: IEEE; 2017:1-6.
34. Salleh S, Marouf S. A learning automata-based solution to the target coverage problem in wireless sensor networks. *MoMM '13*. New York, NY: ACM; 2013:185:185-185:190.
35. Mohamadi H, Ismail AS, Salleh S, Nodhei A. Learning automata-based algorithms for finding cover sets in wireless sensor networks. *J Supercomput*. 2013;66(3):1533-1552.
36. Mohamadi H, Ismail ASBH, Salleh S. A learning automata-based algorithm for solving coverage problem in directional sensor networks. *Computing*. 2013;95(1):1-24.
37. Mohamadi H, Salleh S, Razali MN. Heuristic methods to maximize network lifetime in directional sensor networks with adjustable sensing ranges. *J Netw Comput Appl*. 2014;46:26-35.
38. Mohamadi H, Salleh S, Razali MN, Marouf S. A new learning automata-based approach for maximizing network lifetime in wireless sensor networks with adjustable sensing ranges. *Neurocomputing*. 2015;153:11-19.
39. Alibeiki A, Motameni H, Mohamadi H. A new genetic-based approach for maximizing network lifetime in directional sensor networks with adjustable sensing ranges. *Pervas Mob Comput*. 2019;52:1-12.
40. Mohamadi H, Salleh S, Ismail AS. A learning automata-based solution to the priority-based target coverage problem in directional sensor networks. *Wirel Personal Commun*. 2014;79(3):2323-2338.
41. Mohamadi H, Ismail AS, Salleh S. Solving target coverage problem using cover sets in wireless sensor networks based on learning automata. *Wirel Personal Commun*. 2014;75(1):447-463.
42. Agache M, Oommen BJ. Generalized pursuit learning schemes: new families of continuous and discretized learning automata. *IEEE Trans Syst Man Cybern Part B (Cybernet)*. 2002;32(6):738-749.
43. Zhang X, Granmo OC, Oommen BJ. On incorporating the paradigms of discretization and Bayesian estimation to create a new family of pursuit learning automata. *Appl Intell*. 2013;39(4):782-792.
44. Oommen BJ, Lanctôt JK. Discretized pursuit learning automata. *IEEE Trans Syst Man Cybern*. 1990;20(4):931-938.
45. Lakshmivarahan S. *Learning Algorithms Theory and Applications*. Berlin, Germany: Springer-Verlag; 1981.
46. Najim K, Poznyak AS. *Learning Automata: Theory and Applications*. Oxford, UK: Pergamon Press; 1994.
47. Narendra KS, MAL T. *Learning Automata: An Introduction*. Upper Saddle River, NJ: Prentice-Hall, Inc; 1989.
48. Poznyak AS, Najim K. *Learning Automata and Stochastic Optimization*. Berlin, Germany: Springer-Verlag; 1997.
49. Thathachar MAL, Sastry PS. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Boston, MA: Kluwer Academic; 2003.
50. Yazidi A. Intelligent Learning Automata-Based Strategies Applied to Personalized Service Provisioning in Pervasive Environments [Ph.D. Thesis]; 2011.
51. Oommen J, Misra S. *Cybernetics and learning automata*. New York, NY: Springer Handbook of Automation. Springer; 2009:221-235.
52. Goodwin M, Yazidi A. Distributed learning automata-based scheme for classification using novel pursuit scheme. *Appl Intell*. 2019;50:2222-2238.
53. Chen Y, Zhao Q. On the lifetime of wireless sensor networks. *IEEE Commun Lett*. 2005;9(11):976-978.
54. Mohanty S, Patra SK. Performance evaluation of quality of service in IEEE 802.15. 4-based wireless sensor networks. *Handbook of Research on Advanced Wireless Sensor Network Applications, Protocols, and Architectures*. Hershey PA, United States: IGI Global; 2017:213-248.
55. Yazidi A, Zhang X, Jiao L, Oommen BJ. The hierarchical continuous pursuit learning automation: a novel scheme for environments with large numbers of actions. *IEEE Trans Neural Netw Learn Syst*. 2019;31(2):512-526.
56. Rauniyar A, Kunwar J, Yazidi A, Haugerud H, Engelstad P. Energy efficient target coverage in wireless sensor networks using adaptive learning. Paper presented at: Proceedings of the 2019 Distributed Computing for Emerging Smart Networks (DiCES-N) Workshop. Hammamet, Tunisia; 2019.
57. Gutjahr WJ. ACO algorithms with guaranteed convergence to the optimal solution. *Inf Process Lett*. 2002;82(3):145-153.

**How to cite this article:** Upreti R, Rauniyar A, Kunwar J, Haugerud H, Engelstad P, Yazidi A. Adaptive pursuit learning for energy-efficient target coverage in wireless sensor networks. *Concurrency Computat Pract Exper*. 2020:e5975. <https://doi.org/10.1002/cpe.5975>

## APPENDIX

*Proof of Theorem 1.* The proof follows similar arguments as in Reference 57. Using recurrence, we can obtain a lower bound on  $p_{(i,j)}(t)$ :

$$p_{(i,j)}(t) \geq \prod_{k=1}^{t-1} \lambda_k p_{(i,j)}(0). \quad (\text{A1})$$

Let  $p_{\min}(0) > 0$  a lower bound on  $p_{(i,j)}(0)$ .

Let  $A_t = \{C(t) \neq C^*\}$  the event that at iteration  $t$ , the candidate solution does not contain the optimal solution  $C^*$ .

Let  $B_T$  the event that optimal solution is not found up to instant  $T$ .

$$P(B_T) = \prod_{t=1}^T P(A_t), \quad (\text{A2})$$

$$P(B_T) \leq \prod_{t=1}^T \left(1 - \prod_{k=1}^{t-1} (\lambda_k p_{\min}(0))^m\right). \quad (\text{A3})$$

■

By resorting to  $(1 - u) \leq \exp(-u)$  we obtain

$$P(B_\infty) \leq \prod_{t=1}^{\infty} \exp\left(-\prod_{k=1}^{t-1} (\lambda_k p_{\min}(0))^m\right) \quad (\text{A4})$$

$$= \exp\left(-\sum_{t=1}^{\infty} \prod_{k=1}^{t-1} \lambda_k^m p_{\min}(0)^m\right). \quad (\text{A5})$$

However, from our assumption

$$\sum_{t=1}^{\infty} \prod_{k=1}^{t-1} \lambda_k = \infty$$

Since we have

$$P(B_\infty) \leq 0 \quad (\text{A6})$$

then

$$P(B_\infty) = P(C^* \text{ never obtained}) = 0. \quad (\text{A7})$$

Examples of smoothing sequences which eventually generate the optimal solution with probability 1 (ie, which satisfy the sufficient condition of Theorem 1) includes

$$\lambda_t = 1 - 1/(t+1)^\beta \text{ for } \beta > 1.$$

and

$$\lambda_t = 1 - \frac{1}{(t+1)\log(t+1)^\beta} \text{ for } \beta > 1.$$

Let  $t^*$  the first time instant when the optimal solution is found, the optimal components are always reinforced. For  $t^* + r$ , for  $j$  such that  $j \notin C^*$ , using recurrence, we can verify:

$$p_{(ij)}(t^* + r) = \prod_{k=t^*}^{t^*+r-1} \lambda_k p_{(ij)}(t). \quad (\text{A8})$$

Easy to see from the assumption that  $\prod_{k=0}^{\infty} \lambda_k = 0$ , by considering the log of the expression described in assumption on  $\lambda_k$ .

$$\lim_{r \rightarrow \infty} p_{(ij)}(t^* + r) = 0. \quad (\text{A9})$$

Therefore, for  $j^*$  belonging to the optimal solution.

$$\lim_{r \rightarrow \infty} p_{(ij^*)}(t^* + r) = 1. \quad (\text{A10})$$

*Proof of Theorem 2.* Using recurrence, we know that:

$$p_{(ij)}(t) > \lambda^{t-1} p_{(ij)}(0). \quad (\text{A11})$$



Thus,

$$P(A_t) \leq 1 - (\lambda^{t-1} p_{\min}(0))^m. \quad (\text{A12})$$

Therefore,

$$P(B_T) \leq \prod_{t=1}^T (1 - (\lambda^{t-1} p_{\min}(0))^m). \quad (\text{A13})$$

Thus,

$$P(C^* \text{ never obtained}) = \text{Prob}(B_\infty) \leq \prod_{t=1}^{\infty} \exp(-\lambda^{(t-1)m} p_{\min}(0)^m), \quad (\text{A14})$$

$$\leq \exp\left(-p_{\min}(0)^m \sum_{t=1}^{\infty} \lambda^{(t-1)m}\right), \quad (\text{A15})$$

$$= \exp\left(-p_{\min}(0)^m \sum_{t=0}^{\infty} \lambda^{tm}\right). \quad (\text{A16})$$

Let us define  $h(\alpha) = \sum_{t=0}^{\infty} \lambda^{tm}$ .

$$h(\alpha) = \sum_{t=0}^{\infty} \lambda^{tm} \quad (\text{A17})$$

$$= 1/(1 - \lambda^m), \quad (\text{A18})$$

$$\lim_{T \rightarrow \infty} P(B_T) = P(C^* \text{ never obtained}) \quad (\text{A19})$$

$$\leq \exp(-p_{\min}(0)^m h(\lambda)). \quad (\text{A20})$$

Since we know that  $\lim_{\lambda \rightarrow 1} h(\lambda) = \infty$ , then  $\lim_{T \rightarrow \infty} P(B_T)$  can be made arbitrarily close to zero, if  $\lambda$  approaches 1.

Hence the theorem is proven. Now, let us characterize the LA probabilities at convergence.

Let  $t^*$  the first time instant when the optimal solution is found, the optimal arcs are always reinforced. For  $t^* + r$ , for  $j^* \in C^*$ , we have:

Using recurrence, we can obtain verify

$$p_{(ij^*)}(t+r) = \lambda^r p_{(ij^*)}(t) + (1-\lambda) \sum_{i=0}^{r-1} \lambda^{r-i-1}. \quad (\text{A21})$$

We remark

$$\lim_{r \rightarrow \infty} \sum_{i=0}^{r-1} \lambda^{r-i-1} = 1/(1-\lambda). \quad (\text{A22})$$

Therefore,

$$\lim_{r \rightarrow \infty} p_{(ij^*)}(t+r) = (1-\lambda) \times 1/(1-\lambda) = 1. \quad (\text{A23})$$

■