Energy Efficient Target Coverage in Wireless Sensor Networks Using Adaptive Learning

Jeevan Kunwar

Thesis submitted for the degree of
Master in Network and System Administration
30 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2018

Energy Efficient Target Coverage in Wireless Sensor Networks Using Adaptive Learning

Jeevan Kunwar

Abstract

Day by day innovation in wireless communications and micro-technology has evolved in the development of wireless sensor networks. This technology is used in many application fields such as battlefield surveillance, home security, healthcare supervision and many more. However, due to the use of small batteries with low power this technology faces the issue of power and target monitoring. There is much research done to overcome these issues with the development of different architecture and algorithms. In this thesis, a scheduling machine learning algorithm called adaptive learning automata algorithm(ALAA) is used. It provides an efficient scheduling technique. Such that each sensor node in the network has been equipped with learning automata, and with this, they can select their proper state at any given time. The state of the sensor is either active or sleep. The proposed algorithm results are obtained by doing several experiments. For this experiments, different parameters are used to check the consistency of the algorithm to schedule the sensor node such that it can cover all the targets with the use of less power. The results obtained from the experiments show that the proposed algorithm is an efficient way to schedule the sensor nodes to monitor all the targets with use of less power.

On the whole, this thesis manages to achieve its goal by contributing to the related research on wireless sensor networks with a new design of a learning automata scheduling algorithm. The ability of this proposed algorithm to use the minimum number of sensors to be in active state verified to reduce the use of power in the network. Thus, achieving the goal by enhancing the lifetime of wireless sensor networks.

Keywords—wireless sensor network, learning automata, sensors, targets, machine learning, minimum active sensors, adaptive learning automata algorithm, coverage area

Acknowledgement

First of all, I would like to express my profound gratitude to my supervisors Anis Yazidi and Hårek Haugerud for the enthusiastic encouragement, useful critiques, dedication, and inspiration, which helped me while doing this thesis. I want to take this opportunity to thank them and express how immensely important were their lecture, weekly meeting, a suggestion for the completion of my master's thesis. The positive spirit from my supervisors has always encouraged me to thrive more, work hard and realize my potential.

I would also like to thank Kyrre Begnum for his great lectures and notes on recent technologies in our field of study. That helped me to do my research work and be updated with modern technologies.

I would also like to extend my thanks to my friends and classmates for giving me support, motivation and help in my study period. I sincerely gratitude University of Oslo (UIO)and Oslo and Akershus University College (HiOA) for providing quality education and infrastructures.

I would especially like to thank my family. My wife, Srijana Bishowkarma who has been extremely supportive of me and encouraged me during my study. Thanks for being patient, taking care and loving me always. My parents, who deserve special thanks for their continued support and encouragement.

- Jeevan Kunwar

Contents

List of Figures

List of Tables

Listings

Abbreviations

3D	Three Dimension
ABC	Artificial Bee Colony
ALAA	Adaptive Learning Automata Algorithm
CDS	Connected Dominating Set
CPU	Central Processing Unit
DLA	Distributed Learning Automata
DSN	Directional Sensor Networks
GCSC	Greedy Connected Set Coverage
GPS	Global Positioning System
LA	Learning Automata
LADSC	Learning Automata Disjoint Set Covers
LADSC	Learning Automata Disjoint Set Cover
M	Number of targets
ML	Machine Learning
MLCP	Maximum Lifetime Coverage Problem
N	Number of sensor
RL	Reinforcement Learning

Chapter 1

Introduction

1.1 An Overview Of Wireless Sensor Networks

The development in the field of wireless communication is gradually increasing day by day. Many devices and a lot of software are evolving with an increase in the use of wireless technology. The most trending and popular field of wireless technology is wireless sensor networks. This involves wireless networks consisting of a large number of sensors deployed in a monitored area in a random or deterministic manner and which can be auto-configured. These sensors are small, low power, low cost and multi-functional devices with the ability to communicate at a short distance. The sensors of the network consist of three major components each for sensing, processing and communicating [1]. The evolution of the wireless sensor network technology was first started when developing military applications for monitoring surveillance of the battlefield. The development and innovation of new methodologies in wireless sensor networks have increased the field of its application, and it has been used in monitoring different fields such as home, health care, temperature, disaster prevention, environmental monitoring, pollution, etc.

Sensor networks consist of many embedded hardware devices as shown in Figure 1.1. These hardware devices take part in the operation of the overall sensor network. The following is a brief description of each component:

- **Power Unit:** This unit function is to provide power to all other units of the sensor node.
- **Processor Unit:** The unit main task is to process the sensed data, scheduling the task and control function of other hardware units of the sensor node.
- **Radio Transceiver Unit:** This unit function for doing wireless communication.
- **Memory Unit:** This unit contains a storage part of the sensor node, and it contains both program memory and data memory.
- **Sensor:** This unit is small microcontroller chips which have a small size and can function on small data rates.
- **Geo-Positioning System (GPS):** This unit provides location and time unit of the deployed sensor nodes.

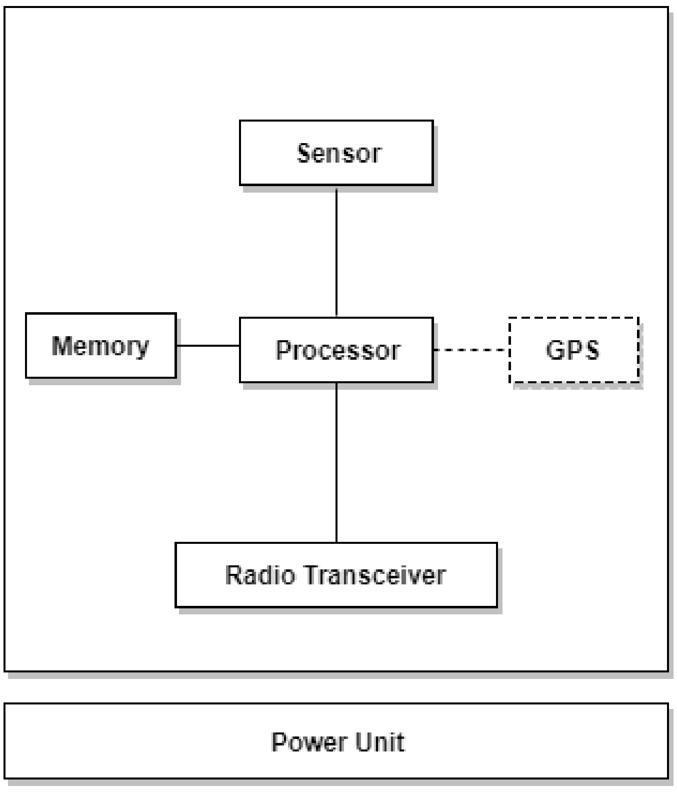


Figure 1.1: Sensor node architecture [2]

1.2 Wireless sensor networks life time

The power management factor is one of the keys concerning factor for the lifetime enhancement of the wireless sensor network. Batteries that are used in a sensor network are relatively small in size and having low power storage capacity. Those batteries need either replacement or frequent recharging for continuous network operation. However, this is impractical for the more extensive and complex sensor networks. These will affect in a network lifetime. This network lifetime defined as the time instant at which the network starts functioning to the time instant where the desired coverage criteria are satisfied [15]. These give the idea that minimization of energy usage results in extension of the lifetime of the wireless sensor network.

Many research papers provide different ideas addressing the problem of inefficient power consumption. They have proposed energy-aware routing, energy-efficient data aggregation and dissemination, transmission power control and nodes activity scheduling for efficient utilization of the energy[21]. These all ideas conclude about maximizing network lifetime by fulfilling all the network coverage requirements.

1.3 Wireless sensor networks coverage Area

Another area of wireless sensor networks is the coverage area. This area can have been defined as the area within which a sensor node monitors and tracks the specified target's activities. Also, it is meant that each target should be monitored by at least one of the sensor node continuously such that there is continuity in the network operation. While doing this, there has been taken care of efficient energy utilization. For this the nodes in the network have been kept in two modes, one is *active*, and another is *sleep* mode. Here nodes in sleep mode do nothing, and no energy is used, but the nodes in active mode monitor their environment or targets. The network lifetime has been enhanced by scheduling the activity of each sensor in active and sleep states [21].

This coverage problem can be divided into three parts. they are as follows [7]:

- **Area Coverage:**
  This coverage includes monitoring of targets in the entire area of the network.
- **Target Coverage:**
  This coverage includes the idea of monitoring only certain targets within the specified region of the network.
- **Barrier Coverage:**
  Barrier coverage includes the idea of minimizing the probability of undetected penetration through the barrier.

There is much research done on coverage problem for designing energy efficient wireless sensor network. For this many scheduling algorithms are proposed to schedule the activity of sensor nodes. One of the scheduling methods for energy efficient wireless sensor network is by using learning automata. This mechanism provides the sensor node to learn their state and select its appropriate state that is active or sleep for maximizing their battery lifetime.

In this, at the initial stage, the assumption has been made that all the sensor nodes are in active mode and they are monitoring at least one targets. So, to make the efficient network and battery usage minimum the sensor nodes are scheduled with active and sleep mode. If the sensor node is not monitoring the target, then it is provided with sleep mode of state, and if it is monitoring the target, the sensor node is provided with an active mode of state. For this concept, the scheduling algorithm using adaptive learning has been proposed. Where, each node are equipped with a learning automata by which each node selects its state to one of the modes, Active or sleep during the network operation. In sleep mode, a node does not use energy, and when it is in the Active mode, it uses its energy.

Figure 1.2, shows the Venn diagram of sensors and 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 color represents coverage of each sensor. The bipartite graph shows the relationship between sensors and their number of the covered targets.

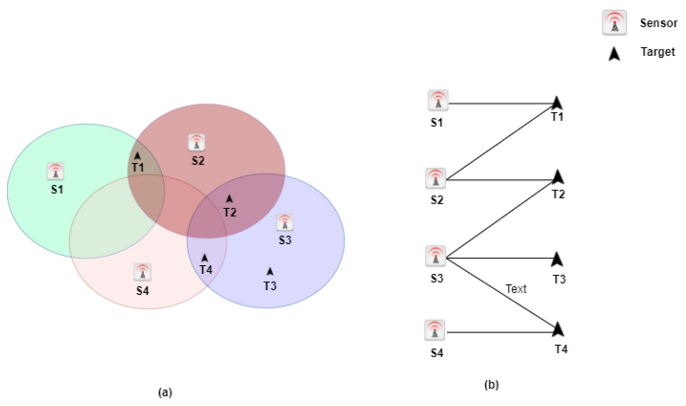


Figure 1.2: (a)Sensors coverage with their targets and (b) Bipartite graph of sensors and targets

1.4 Problem Statement

How to achieve energy efficient target coverage in a wireless sensor network using adaptive learning

Let us consider that there are a set of M targets denoted by $M = M_1, M_2, ...M_m$ which are monitored by the set of sensor nodes N denoted by $N = N_1, N_2, ...N_n$ and these two sets are deployed in an X×X area such that each sensor node covers all targets. All sensor nodes have a fixed sensing range "R" in each experiment. Also, there has been assumed that there are more sensor nodes than targets. It is because to provide continues monitoring of targets in the sensor networks. That helps in conserving energy and maximizing network lifetime. To schedule, the sensor nodes scheduling algorithm has been proposed. That will provide the state of sensor nodes. There are two states of sensor node one is Active, and another is Sleep. If the target is within the range of the sensor node, then its state is active. So, if the target is beyond the range of the sensor, then it is in a sleep state. In an Active state, the sensor consumes some amount of energy while in sleep state it does nothing. A target point M_j within the range 1<=j<=M, is said to be covered by the sensor node N_i if it falls inside the range of one of the sensor nodes 1<=i<=N [7].

The following notations are used in the project:

- N, number of sensors
- M, number of Targets
- X×X, area for deploying sensors and targets
- R, Sensing range of the sensor node
- Γ, active time period of both sensors and targets at initial state
- N_i, be the i^{th} sensor node in 1≤ i≤ N
- M_j, be the j^{th} target in 1≤ j≤ M

The main objective is :

- To schedule the sensor node activity so as to achieve maximum network lifetime.

For the sensor deployment there a set of M targets $M = M_1, M_2, ..., M_m$ and N sensor nodes $N = N_1, N_2, ..., N_n$ are taken in a X×X region. To achieve a maximum lifetime sensor are deployed redundantly in such a manner that each sensor can monitors at least one targets.

For the sensor scheduling, adaptive learning automata algorithm has been used. This method will help sensor nodes to select their state according to their probability vector without affecting the operation of the network. This learning automata algorithm helps the sensor nodes to select their appropriate state according to need of operation of network.To find the target coverage, first euclidean distance between sensor node and targets has been calculated. Comparing euclidean distance with sensor sensing range, the sensor checks the target within their range. If the calculated Euclidean distance is less than sensing radius "R" then the sensor is monitoring the target. At this time Sensor selects its state to active state. Otherwise, it will be in a sleep state and remain inactive. In the active state, sensor consumes some amount of energy and in a sleep state, it consumes no energy.

After many iterations, proposed algorithm will select best active sensor nodes among all the active nodes that are covering the maximum number of targets. And this helps in minimizing the number of active sensors. This means that by the use of less number of active senors there is use of less energy and thus overall network energy is conserved and it leads to lifetime maximization.

This project will be solving following problems:

- How to observe the number of sensors that are active?
- How to schedule the sensor node for efficient energy conservation of the wireless sensor network?

1.5 Report structure

- Introduction - section describes the problem and challenges of the thesis.
- Background - section includes basic theory and technical terms that are used in the project.
- Approach - includes the approaches that are used to solve the problem statement.
- Design - This section of the report contains Design of the project environment setup and algorithm used in the project.
- Implementation And Result Analysis - This part of the report contains about design implementation and result obtained.
- Discussion - part contains how precise our result is obtained and also about the unsolved tasks.
- Future work and conclusion - part contains what can be done to improve and new ideas for improvement. Along with the short conclusion stating about what has been obtained and what went wright and wrong.

Chapter 2

Background

2.1 Wireless sensors

In the present day, there is rapid progress in the field of wireless communication. This field consists of many sub-fields among which one of the sub-fields is wireless sensor network. This wireless sensor network includes of the sensor devices, which are equipped with one or more sensors, one or more transceivers, processing storage resources, and possible actuators. This device collects, stores and process the environment information where they are deployed. So sensor network is densely implemented with limited resources and dynamic topology. Due to the limitation in size and weight of the sensors, there is the main issue in the power scarcity problem. Power utilization of the sensor network is directly related to lifetime enhancement and application performance of the sensor network. So there is essential to optimize the energy used by sensor network operation. Recently many research works have been done in the field of wireless sensor network energy optimization.

To develop the sensor network as energy efficient network, power management of sensor nodes can be done as follows:

1. Scheduling the nodes by using active and sleep mode.
2. Managing transmission range between the wireless nodes.
3. Using energy efficient routing and data gathering methods.

In this paper, the main focus has been on the node activity scheduling method for target coverage problem. This method provides an efficient result with a randomly deployed sensors that are monitoring the fixed list of targets. There is some performance requirement such as routing connectivity, network coverage, redundancy requirement, etc. which are used for scheduling nodes. For the target coverage problem, here assumption has been made about that each node radio range is capable of maintaining their route. So each target in the network is covered by more than one sensor nodes where the redundant nodes are put into sleep state without affecting the coverage.

In the past, many papers provide the idea of scheduling by dividing the sensor nodes into disjoint subset covers by assuming that at some instant time unit one set cover is active for monitoring the targets. This leads to the problem of disjoint set cover problem.

This paper provides the idea of using the learning automata for scheduling the activity of the sensor nodes without dividing them into disjoint sets. In this, each sensor node is equipped with learning automata such that each node can select their proper state whenever they need with the requirement of the network operation.

2.2 Machine learning

The Machine learning (ML) was introduced in the late 1950's as a technique for artificial intelligence (AI)[4]. Due to diversity in the field of machine learning application. There are many ideas evolved to apply this technique in different fields of computer science. There is extensively use of machine learning techniques from past. Many task can be done using the machine learning. For example it can be used in performing task like regression, classification etc. Application area of computer science like for speech recognition, biometric identification, computer vision, image processing and many more are using machine learning techniques. Many filed like mathematics, neuroscience, computer science are used to develop machine learning algorithms.

The importance of machine learning is[3]:

1. To find the answer to the issues of knowledge forms and increase execution of developed systems by the use of machine learning models.
2. To adopt a computational model for improving machine operation. For this training data set are analyzed with their pattern and consistency.

The following are the types of machine learning algorithms

2.2.1 Supervised Learning

It is a predictive model.[5] Where, it provides the relationships and dependencies between input, output and the system parameters. Based on this relationship one can predict the output values for new data which it learned from the previous datasets. This learning section includes a classification and regression problem. [3]This learning algorithm is used to solve many challenges in wireless sensor network such as media access control, security, and intrusion detection, localization and object targeting, data integrity and fault tolerance, etc.

2.2.2 Unsupervised Learning

Learning has been done by the unlabeled test data. This type of learning is used in pattern detection and descriptive modeling. By the use of pattern and rules, this learning derives meaningful understanding to the data to the user. It includes Clustering and Association rule as a main unsupervised learning algorithms [5]. The basic idea of this type of learning is to classify data into a group of similar sets.

2.2.3 Reinforcement Learning

This machine learning maximizes its performance by allowing its agent to select state automatically form environment.[5] Agent learns its behavior through the use of simple reward feedback signal. It uses a specific type of problem and its solution to provide a Reinforcement learning algorithm. There is much application of reinforcement learning algorithms. Some of them are computer played board games (Chess, Go), robotic hands, Q-Learning, and self-driving cars, etc.

Machine learning in a wireless sensor network can be used for creating a prediction model by the use of machine learning tools and algorithms. So also by understanding themes and patterns of machine learning, users can implement it in a wireless sensor network to get more flexibility.

2.3 Learning Automata

Learning automata are the model of machine learning. This model is the abstract model that selects the appropriate action among the finite set of actions and performs it in a random environment. Then with the help of the reinforcement signal environment evaluates the selected action and responds to the automata. To select the next action, automata first update its internal state from selected action and signal is received. Thus, by following certain rule automata finds the best output solution and to do this automaton continuously interact with the environment to make the appropriate decision for suitable action.

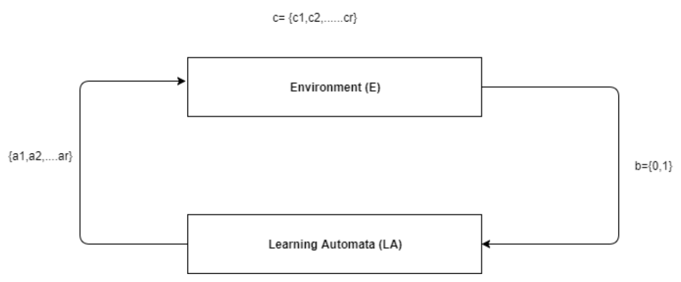


Figure 2.1: Projection of Learning automata and random environment association [21]

Environment can be defined by the triple E = (a, b, c) where a, b, c can be defined as follows: a = a_1, a_2, … , a_r: represents a finite input set b= b_1, b_2, … , b_r: represents the output set and c = c_1, c_2, … , c_r: is a set of penalty probabilities

Each element c_i of c corresponds to one input of action a_i. An environment contains three models. They are as follows:

- P-model:
  This is the environment in which b can take only binary values 0 or 1
- Q-model:
  In this model, the environment allows finite output sets with more than two elements that take values in the interval [0, 1].
- S-model:
  In this model, the output of the environment is a continuous random variable which assumes values in the interval [0, 1].

Also, learning automata are classified into fixed-structure stochastic and variable-structure stochastic models.

[35]A learning algorithm can be defined as follows:

$$p(n+1) = T[p(n), a(n), b(n)] \tag{2.1}$$

Let $a(k)$ and $p(k)$ denote the action chosen at instant k and the action probability vector on which the preferred action is based. The repetition equation shown by 2.2 and 2.3 is a linear learning algorithm, which is used to update the action probability vector p. Let $a_1(k)$ be the action chosen by the automaton at instant k.

$$\begin{aligned} p_i(n+1) &= p_i(n) + a[1 - p_i(n)] \\ p_j(n+1) &= (1-a)p_j(n) \forall j, j \neq i \end{aligned} \tag{2.2}$$

when the taken action is rewarded by the environment (i.e.,$b(n) = 0$)and

$$\begin{aligned} p_i(n+1) &= (1-b)p_j(n) \\ p_j(n+1) &= \frac{b}{r-1} + (1-b)p_j(n) \forall j, j \neq i \end{aligned} \tag{2.3}$$

when the selected action is penalized by the environment (i.e.$b(n) = 1$)
Here, r is the number of actions taken and a and b denote the reward and penalty parameters and determine the number of increases and decreases of the action probabilities.The parameter $p_i(n)$ and $p_j(n)$ are the probabilities of action a_i and b_i [51]. [35] For a = b, learning algorithm is called Linear Reward-Inaction(L_{R-I}) algorithm,for b «a, it is called Linear Reward epsilon Penalty($L_{R-\epsilon P}$)algorithm, and for b = 0, it is called linear reward–penalty(L_{R-P})algorithm .

2.4 Tools used

Python programming language has been used in a project for the programming part and the simulation part there is the use of Matplotlib module of python program.

2.4.1 Python

Python[28] is a high-level programming language for general purpose programming. It has a big library supporting different programming types. It is easy to work with and also provide a great variety of possibilities. It helps the programmer to work quickly and efficiently integrate the system. It has an interpreter that support a large number of operating systems. It also supports to develop many graphical user interfaces. It has a feature like dynamic system and automatic memory management capabilities. These features make it more popular among the programmer. There is an observation of who the clear leader is, python. It is one of the reasons for using Python as the main programming language in this thesis. The philosophy of Python is simple, as described in the document Zen of Python[28], including the following lines:

- Beautiful is better than ugly
- Simple is better than complex
- Complex is better than complicated
- Readability counts
- Errors should never pass silently

2.5 Related work

Table 2.1 Related Paper

Year	Authors	Issued Focused	Approach Used
2004	Chi-Fu Huang	coverage in 3D space	Efficient polynomial time algorithm
2005	Michaela Cardei	Disjoint set cover	Linear programming and greedy approach
2005	DING-ZHU DU	maximum disjoint set covers problem	MC-MIP (Maximum Coverset-Mixed Integer Programming)
2007	Yingshu Li	k-coverage Schedule	PCL-Greedy Selection (GS) and PCL-Greedy Adjustable(GSA) algprithm
2007	S.Mini	Sensor deployment and Scheduling	Artificial bee colony algorithm particle swarm optimization for sensor deployment and Heuristic for scheduling
2008	Ionut Cardei	Connected set cover	Integer Programming, Greedy approach and Distributed and Localized heuristic
2010	Maggie X, Cheng	k-coverage	Linear programming based exponetial time exact solution and approximation algorithm
2012	Guofang Nan et al	network life time and network coverage	Coverage-guaranteed Distributed sleep wake scheduling (CDSWS)
2014	Habib Mostafaei	target coverage and maximum life time	LADSC algorithm
2015	Jiang Lei Shu	coverage problem of industry wirelesssensor netowrk	Communication weighted greedy cover, Optimized connected coverage heuristic, Overlapped target and connected coverage and Adjustable range set covers.
2018	Dayong Ye et al	saving energy of each node	Self-adaptive sleep-wake-up scheduling

In [22] there is an explanation about management of target coverage problem. That mentioned that there is a need to manage targets to get a maximum lifetime. To do this paper has used a probabilistic coverage model which takes the distance parameter for target coverage. This algorithm is based on the modified weighted set cover. Which helps in organizing sensors into disjoint and non-disjoint set covers.

In a wireless sensor network, there is a problem of selecting an appropriate node that is covering targets. This concept of coverage-centric node selection problem is formulated in [40]. To get the solution of the problem authors have used coverage-centric active nodes selection (CCANS) algorithm. This algorithm is based on the formation of the connected dominating set (CDS). The active nodes of the network form the connected dominating set. This provides the backbone to other nodes for sensing and communication purpose. Such that sensor node data communication is processed through this route.

In [39] authors have discussed applying the sleep and awake schedule to the low duty cycle wireless sensor networks. In this, they have taken in consideration of the explicit effect of synchronization error for designing sleep and awake schedule. To do this, they have divided their work into two parts. The first part of the work provides sleep and awake schedule by use of an efficient search method. Such that there is the use of minimum number of sensors for coverage. So in the second part, they have mentioned finding the quality of service of the network as a whole.

To enhance the lifetime of the wireless sensor network. Have to schedule the activities of cover sets. These cover sets are the sets of sensor nodes that are covering the number of targets. So for obtaining this cover sets, sensor node should be deployed in a proper manner such that target coverage is achieved. Authors in [19] provided heuristic and artificial bee colony algorithm to find the network lifetime. So for sensor deployment, authors proposed two scheduling techniques, heuristic, and an ant colony. By their experiments, authors provide that their methods help in improving the network lifetime of the sensor networks. In [21] there is the explanation of the solution to the cover set problem as mentioned in the above paper. Here the authors explained the learning automata algorithm. This algorithm is used to schedule the sensor activity of the nodes such that the network lifetime is maximized.

There is much literature that can be found about the target coverage problem of the wireless sensor networks. In [45] authors have discussed the target coverage along with data collection problem in wireless sensor networks. This data collection help for transmitting sensed data from nodes to the sink. By the use of polynomial-time approximation and polynomial-time constant approximation methods, the problems as mentioned earlier are analyzed. Experiments are done by taking sensor with same sensing radius and transmission radius. So the result of the paper show that it is NP-hard to find maximum lifetime by scheduling target covers and data collection in wireless sensor networks.
The sensor communicates with its neighbor node to transfer information with each other. To do this sensor use some amount of energy. However, for high data rate transmission, it requires a large amount of energy. So this issue of the sensor network should be considered for a network lifetime. For this [46] provides the energy efficient pattern for high data rate communication in wireless sensor networks without affecting the rate of sensing coverage.

In [6] the sensor nodes are organized into several maximal set covers. These set covers are activated to monitor the targets, and other nodes remain in sleep mode to save the energy. The main goal of the paper was to find the disjoint set for energy conservation of the sensor network to increase the lifetime. they have used two-heuristic approach for computing the sets by using the linear programming

and a greedy approach. The result shows that there is an increase in a lifetime with an increase in target and sensing range with a specified number of targets. So if there is a decrease in the number of targets, there is an increase in the number of lifetimes.
A heuristic method for organizing sensor nodes into disjoint set covers is mentioned in [32][23]. These disjoint set covers are activated successfully. Such that the sensor set covers that are an active state only can monitor the targets and other sensor sets go to low energy sleep mode. Also there use of Greedy connected set coverage (CSC) heuristics algorithm in [32]. This method helps in increasing the network operation lifetime in comparison to IP-CSC heuristic algorithm.

To obtain adequate target coverage sensors can be divided into cover sets. This cover set is provided with some interval of time to monitor targets. However, this solution is proved to be NP-complete. So to obtain coverage solution authors in [26] has proposed for the centralized heuristic algorithm. This algorithm generates cover sets that help in monitoring all the targets.
If there is a need to deploy the sensor nodes in a large area like for monitoring the environmental risk. For this, the large number of sensor nodes has to be deployed. So proper monitoring algorithm is needed to get better throughput. There is a physics-based heuristic algorithm described in paper [38]. This algorithm helps in placing the scalable sensor node efficiently to find the coverage of fixed targets in wireless sensor networks. Paper has introduced the concept of virtual sensors. This is used for moving and merging the sensors to get the minimum number of target covering sensors.
To know the answer of question how to manage the schedule of a large amount of sensor node deployed can be obtained in paper [41]. This paper provides the large sensor deployment as the maximum lifetime coverage problem(MLCP). So by the use of polynomial-time approximation algorithm authors provide the solution to the mentioned problem.

In [11] k-coverage problem in a wireless sensor network is highlighted. This k-coverage defines the user-defined area of sensor coverage where the sensor can cover its targets. For this two algorithms were formulated in the paper. First one is PCL-Greedy- Selection (GS) algorithm which deals with the sensors that have fixed sensing range and belongs to disjoint subsets. Next is PCL-Greedy-Selection-Adjustable (GSA) algorithm which deals with the sensor that belongs to non-disjoint subsets and sensors with adjustable capabilities of their sensing range. Comparing both algorithm paper concluded that the GSA algorithm is a better algorithm to solve the k-coverage problem in wireless sensor networks.

Two algorithms linear programming-based exponential-time exact solution and an approximation algorithm in [10] are used for maximizing the lifetime of sensor networks. In linear programming based exponential time exact solution, first the non-redundant set cover is obtained, and then the schedule for each non-redundant set cover is computed. In approximation algorithm first, k-covers are discovered. This K covers are the number of non-redundant set covers that include most of the targets. Then the lifetime optimization is done by using linear programming. To obtain k-coverage in dense sensor networks authors in [14] has purposed an efficient approximation algorithm that finds the coverage space within the logarithmic optimal. This distributed algorithm provides the best solution to node optimization and helps to prolong the network lifetime.

How appropriate coverage strategies can be chosen for better performance of the wireless sensor device in the industry has been addressed in [24]. For target coverage problem, this paper proposed four algorithms. These four algorithms were analyzed based on their network lifetime, coverage time, average energy consumption, etc. Results are observed to select the appropriate algorithm to optimize the coverage and connectivity of industrial wireless sensor network.

Heuristic scheduling for the sensor nodes and Artificial bee colony (ABC) algorithm with Particle Swarm Optimization(PSO) for sensor deployment problem is formulated in the paper [15]. By Comparing both purposed algorithm author concluded that to find the optimal locations the Artificial Bee Colony is a good choice and for obtaining a maximum lifetime of the sensor network heuristic scheduling is appropriate.

There is a connected set cover problem in wireless sensor networks. In [25] there is a detail description about it. The goal of this paper is to find a maximum number of set covers. Where activated set is connected to the base station. The objective of the paper is fulfilled by using three solutions as an integer programming based solution, a greedy approach, and a distributed and localized heuristic.
There is a combination of two coverage and connectivity problem of wireless sensor networks into one problem in paper [48]. To do this, each sensor node is provided with a priority. For this, there is a use of Efficient Energy Coverage and Connectivity(ECC/EC2) novel algorithm. Which is used to configure the wireless sensor networks dynamically? And the result of the algorithm provides a degree of coverage and connectivity. Also in [9] there is mention of the cover set problem. This problem is formulated as a maximum cover set problem by using the proposed Imperialist Competitive Algorithm (ICA) approach. This approach is used to schedule the deployed sensor nodes such that they can monitor all the targets in the network with maximizing the network lifetime.

In wireless heterogeneous sensor networks, there is also a coverage and connectivity problem. To solve this problem in [12] authors provide two heuristic solutions.Which are Remaining Energy First Scheme (REFS) and Energy Efficient First Scheme (EEFS). For formulating these schemes paper first state the problem as a connected set cover problem and then it states it as integer programming constraints.

In Directional Sensor Networks (DSN), there is a problem of designing the appropriate algorithm which provides coverage of all targets and their connectivity to the sink. The paper [17] provides the idea of using the distributed learning automata (DLA) algorithm to solve this problem. The author mentioned that this algorithm schedule the sensor nodes to be active at each stage to cover all the targets and send the sensed information to the sink node.

For maximizing the network lifetime of the wireless sensor network in [18] there is the highlight of the critical target and critical sensor aspect. This critical target is the targets that are covered by least number of sensors, and this sensor is called critical sensors. To enhance the lifetime of the sensor networks author proposed heuristic for selecting a minimum number of critical sensors. Also, this sensor will cover critical targets for a longer time thus by including all the targets.

Paper [16] provide the answer to a question about how to monitor the coverage problem in 3D space. The proposed solution is to use the efficient polynomial-time algorithm. The coverage problem of sensor networks is formulated as a decision problem. So all the sensor coverage area is monitored to find if all the targets fall within the sensor nodes range.

In the case of the mobile target, it is difficult to find the exact coverage and position of targets in large-scale wireless sensor networks. There is a trap model for the solution. However, in real time Wireless sensor networks it is difficult to implement. In a practical scenario, sensor sensing follows probabilistic sensing mode. In [20]authors has proposed a probabilistic sensing model and circular graph for detecting the mobile targets. They formulated probabilistic trap coverage with maximum network lifetime problem. So circular coverage graph for determining that if given sensor networks can provide the probabilistic trap coverage or not.

2.6 Proposed Algorithm

The purpose algorithm in this paper is the Adaptive learning automata algorithm for scheduling the sensor nodes. This algorithm helps in finding the best active sensors that are monitoring the maximum targets. The flow of the algorithm is divided into three phase which includes the initial stage, a learning phase, and the target monitoring phase. The initial period starts with broadcasting the message containing the information of the sensor node to its neighbors. It ends with reply information from a neighbor and knowing the monitored targets. Then the learning phase starts with selecting the state of a node by using learning automata and ends by providing the appropriate probability vector value for the entire operation. The final stage is the target monitoring phase — this starts with selecting the best action of sensors using learning automata and ends by providing a sensor to operate according to this best action. Each stage is described briefly in the following part.

2.6.1 Initial Phase

In this phase, each node is equipped with the learning automata LA_i. Which help Sensor node to select their state to active or sleep. At first, each state of the sensor node is provided with equal 0.5 probability. Then the sensor node starts communication with its neighbor nodes by broadcasting the message. This message contains id, number of neighbor it covers and its position. Then it waits to receive a reply from the neighbor nodes. This network operation divided into rounds which contains learning phase followed by target monitoring phase.

2.6.2 Learning Phase

This phase includes sensor nodes with learning automata. This learning automaton helps the node to select one of the states from Active or sleep. Then node sends its state message to its neighbors. It waits for a reply from its neighbors. After receiving the reply sensor node do the following:

1. If LA_i is active:
   If its neighbor nodes cover the targets of the sensor node under its coverage, then the sensor node decrease its probability of covering target by using learning automata.
2. If LA_i is sleep:
   If its neighbor nodes also cover the targets of the sensor node under its coverage. Then the sensor node increases its probability vector using learning automata.

This learning phase continuous till the threshold value is meet and operation exceeds the maximum time.

2.6.3 Target Monitoring Phase

In this phase, the sensor nodes select its state for the whole operation as an active or sleep. For this, it selects from the higher probability action of the probability vector of its learning automata if the probability value of active state is high then the sensor selects its state as an active state and vice versa. Also in this phase, the sensor monitors the target until it loses its all energy.

Chapter 3

Approach

The main focus of this chapter is to formulate an answer to the problem statement. This chapter is divided into many steps, and each step gives a solution to add one step forward to our projected goal.

3.1 Objective

The main objective of this thesis is to maximize the network lifetime of a wireless sensor network by using the learning automata algorithm.

3.2 Experimental Environment

The experimental setup is done first by installing Pycharm integrated development environment (IDE) in the system. This works as a front end for developing the program. Different modules that are needed in the program are installed in this environment. This is used as front-end to develop and analyze the code of our project. For plotting graph of our result, there is the use of python GUI modules as the graphical interface. This graph is used to analyze the results in the end. To make this algorithm work many resources of the environment are used.
The following table shows the physical specification of the device in which the environment is setup:

Table 3.1 Physical specification of system

CPU:	Intel(R) Core(TM) i7-6500 CPU @ 2.50GHz
Cores:	4
OS:	Windows 10 Home 64-bit
Memory:	8GB

3.3 Technologies

Python programming script used for developing the workflow of the project. The machine learning algorithm is implemented to schedule sensor nodes to select their action of states. There is the use of draw.io to draw some diagrams in this project. So for writing this report overleaf has been used and to do proofreading of the writing part, Grammarly tool is used.

3.4 The Plan

Planning is important to reach every goal. So, this project work is planned into many phases that are according to the requirement for achieving the goal of this project work. So this work will be completed on given time instance.

At first, research and study of the project related work have been done. This phase is of finding the related work as mentioned in our background chapter. Here different approach that has been for solving problem-related to our task is found out. With this research, there is a finding of the answer to our question about how different paper has implemented machine learning in wireless sensor networks.

When the above task completed then phase the second start to create an environment for the project. Here the environment means physical as well as the technical part of the project.

The third phase of our plan starts with the implementation of an algorithm for the project. In this project, the adaptive machine learning algorithm is used. This algorithm help sensor nodes in our project to learn automatically about their Action of their states. This states are Active and sleep state. Active state, the sensor is monitoring a target, and in a sleep state, it is doing nothing. So the sensor is using some amount of energy during Active state.

In the fourth phase, the results are obtained by the running experiments the of the project. These experiments are done by varying the parameter of the project to obtain results. There is a plan of doing five to seven experiments in this project. Such that the potentiality of our implemented algorithm can be easily observed.

Finally, obtained results from the above case are evaluated and analyzed by using graph and tables. These results are then discussed in detail in the discussion chapter of the project. So later conclusion was made from the results obtained.

3.5 Planned Experiments

Planned experiments help in executing the solution of the project. Only the experiments that are done in a proper setup will give valid results. To do this, there should be the appropriate setup of the environment, interfaces, resources available, etc. Such that algorithms can be implemented and obtained results from the experiment can be tested thoroughly.
In this project, there is N_sensor and M_Targets deployed within the area of 600m$\times$600m. Sensors deployed in each experiment has the same sensing range. To schedule the sensor nodes, there is the use of learning automata algorithm. The parameters that are used during the experiment period are defined as follows:

N_Sensor, It represents a total number of sensor that is using in the project. In this project, between 4 and 90 sensor nodes are deployed to check the functionality of the purposed algorithm.

M_Target, it represents the total number of targets to be covered by the sensor node. In the project, the number of targets deployed is between 3 and 60

So to represent a range of the sensor, Range_sensor, It represents the sensing range of the sensor. Its value ranges between 50 and 600.

LA is the learning automata algorithm that is used in the project. This helps the sensor node to select their state. There are two states one is active, and another is sleep.

3.5.1 Experiment one

Goal: Focus is on finding a minimum number of active sensors covering all targets by using different values of the sensing range of sensors.

3.5.2 Experiment Two

Goal: Focus here is to check the effect of the varying number of sensor, with fixed sensing range to get minimum active sensors.

3.5.3 Experiment Three

Goal: To find an impact of target density in finding minimum active sensors.

3.5.4 Experiment Four

Goal: To check the efficiency of proposed learning automata algorithm. This is done by varying the parameter of the purposed algorithm. Here, λ of the learning automata algorithm is varied by residing ϵ constant within the entire operation.

3.5.5 Experiment Five

Goal: To check the validity of the experimental results by creating a 2^n binary combination of sensors where n is the number of sensors that are used in the experiment. Which is called a brute force method. For example from 3.5.1 the value of n=10 is taken. So there will be 1024 possible binary combination of 10 sensors. From this combination, it extracts only the combination of active sensors that are covering all targets. This is used to check the combination obtained is correct or not. This brute force method creates all possible combination list of active sensors. This list is compared with a previously obtained list for checking the output of the proposed algorithm.

3.5.6 Experiment Six

Goal: To evaluate the effect of density of sensor and target on learning automata operation.

3.5.7 Experiment Seven

Goal: To compare the efficiency of proposed learning automata with others work.

3.6 Results of the experiments

The results from the above experiments provide an answer about how efficient is our purposed algorithm to meet our problem statement. This results of the experiments are briefly described in the Implementation and result from the analysis chapter of this thesis paper.

3.7 Constraints of the project

In project work, some constraints create a problem in the completion of the project on time. This constraint may be due to time, environment set up or may be due to some technical error, etc. This constraint should be taken into consideration while doing project work.

3.7.1 The Setup

The setup constraints come when there is to modify the algorithms.

3.7.2 The time

A lot of research work is done before starting any project work. Such that the authors can get some idea to do their project however If there is no idea of how to search those related works and extract the required information. Then it will difficult to do the project author to complete on time. So sometimes there is difficulty in the technical part of the project which consumes time to find its solution. So time constraints are a more challenging aspect to take into mind while doing any project work.

3.7.3 The Technology

In the project python programming language is used. This programming language is easy to implement. Many python modules that are needed in this project can be easily used. However, sometimes there is difficulty to use the modules of python to get desired results. It is due to the mismatch of a version of the python interpreter that has been used and sometimes with logical errors.

3.8 Other Implementation

This includes the remaining part of the project that is used in the project. This part describes the portion of the project that is not described in the above part. These elements of the project help in doing the above experiments thoroughly.

3.8.1 Script

The script of the project is developed by using the Python programming language. For this Pycharm is used as the Integrated Development Environment(IDE). This script is divided into four parts for simplicity.

Script For Generating sensors

This part of the script is used to generate the number of sensors for the project. Sensors are randomly deployed. So there is also a use of a script to check redundancy of deployed sensors. Sensors are initially provided with their range. Here the assumption is made that sensor interact with each other automatically. In every iteration of the program, there is a change of sensor orientation because of its randomness.

Script For Generating Targets

This part of the script generates the target of the project. Target is also randomly deployed. So, redundancy check is formulated to avoid the duplication. In this project, there is always less number of target deployed than sensors.

Script For Implementing Algorithm

Here the script for purposed learning automata algorithm is included. This part of the script is used to schedule the sensor nodes to cover the target. So that it provides the minimum number of active sensors where these active sensors are the sensors that are covering targets. The probability vector of the active sensor considers them for the next iteration. From the list of active sensors at the end, the algorithm provides the best active sensors which are covering all the targets of the network.

Script For Comparing Purposed Algorithm With Other Work

This part is used to evaluate our purposed algorithm results with the related works algorithms.

3.9 Expected results

From our experiments and related work, the expected results can be assumed. These results are obtained by doing tests. In this project, sensor, target and learning automata have been used for formulating the problem statement of the project. So the obtained results are analyzed and compared them with other related works to find more fairness of our algorithm.

Chapter 4

Design

4.1 Overview

This chapter contains more detail about the environment setup an algorithm that is used in this project. How design is implemented and formulated according to the plan in approach chapter is going to be explained here. By the use of figures, tables, and some scripts the design is more simply explained.

In this chapter, the design phase is divided into small subparts. This part is used to explain the plans mentioned in the approach chapter. Which contains about what are the circumstances es that is used to make the project complete. There is an explanation about what standards and rule to take for the designing our project according to the flow of plan.

4.2 Sensor Deployment

This includes a description of how sensors of the project are deployed and what strategies are taken for this. There are many sensor nodes deployment models such as uniform random, square Grid and Tri-hexagon tiling [49]. In his project uniform, the random model is used. Such that the sensors are deployed randomly in a rectangular area with the same sensing range. Here, the assumption of a number of sensors is always greater than the number of deployed targets is made. At the initial state, all sensors have an equal probability value of covering the targets. The use of the proposed learning automata algorithm helps the sensor node to learn their state automatically in each iteration depending upon its state the probability value changes. The use of distance formula calculates the coverage of the sensor network. This formula provides the distance between the sensor node and the target of the network. If this obtained value is less than or equal to sensor sensing range, then the sensor is covering the target. Otherwise, the target is out of range of sensor node.

The following is a small script for deploying sensor node in network randomly:

```
import random
from math import sqrt
import matplotlib.pyplot as plt
gridsize=600
Range_sensor= 100
N_sensors=20
```

```python
M_Targets=10
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets

for i in range(N_sensors):
    x_sensor[i]=random.randint(0,gridsize)
    y_sensor[i]=random.randint(0,gridsize)
    print("sensor", i, "is at position", x_sensor[i],y_sensor[i])

    x1 = x_sensor[i]
    y1 = y_sensor[i]
    plt.text(x1 * (1 + 0.01), y1 * (1 + 0.01), i, fontsize=12)  # code for ploting the points with text

    plt.scatter(x1, y1, 100, 'g', '1')  # for scatter plot of sensors
    circle = plt.Circle((x1, y1), Range_sensor, color='black',
                        fill=False)  # this code is for drawing circle coverage of sensor
    fig = plt.gcf()
    fig.gca().add_artist(circle)
plt.ylim(0, gridsize)
plt.xlim(0, gridsize)


plt.xlabel('x - axis')
    # naming the y axis
plt.ylabel('y - axis')
    # giving a title to my graph
plt.title('Sensors')

    # show a legend on the plot
    #plt.legend()

# function to show the plot
plt.show()
```

Listing 4.1: Sensor Deployment code

Here, in the above Listing 4.1the variable `N_Sensors` is the number of sensors that are going to deploy. For example, in this script `N_Sensors` value is 20 with sensing range 100 and a grid size of the network is 600m×600m. This value of the sensor varies with the need of the experiment. The position of the sensor is given by `x_sensor[i]`and `y_sensor[i]` variable. This is the position of the sensor inside x-axis and y-axis. To visualize this code in the graphical form, there is the use of python graphical module called matplotlib.pyplot. The following Figure 4.1 shows the graphical representations of deployed sensors with their coverage area.

In this Figure 4.1, green towers are the sensor and the black circle represents the coverage area of each sensor with a sensing range of 100m.

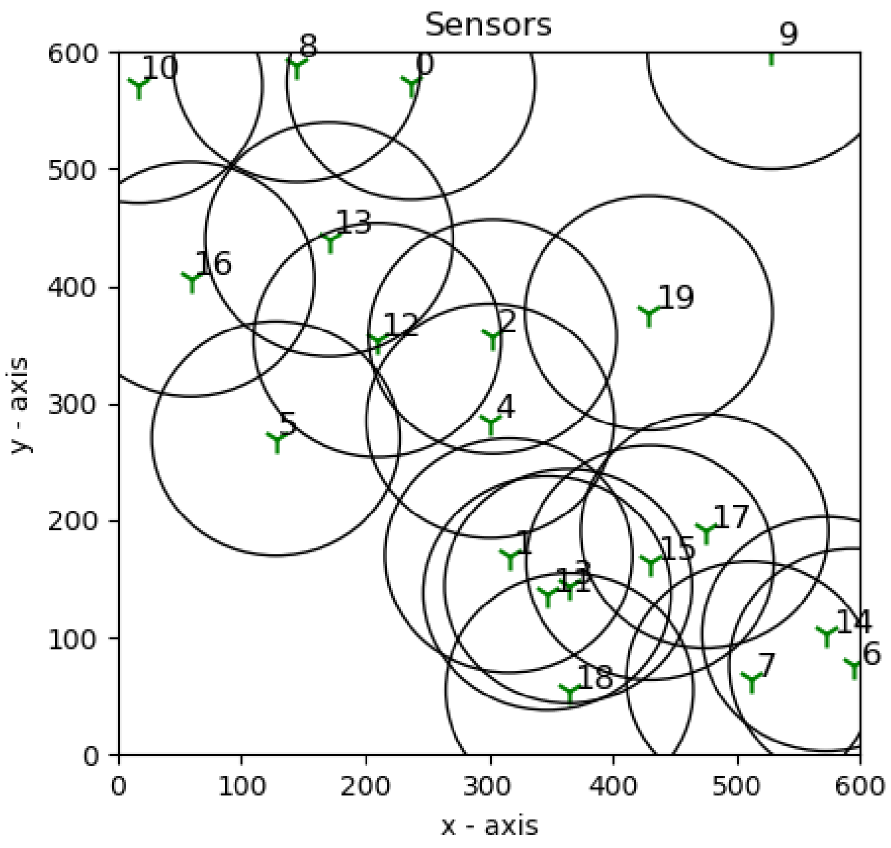


Figure 4.1: Sensor deployment with their coverage area

4.3 Target Deployment

The target is deployed randomly as sensors in this project. Which have a random position? The number of the deployed target is always less than deployed sensors. Because of the assumption that defines there should be at least one sensor that should cover the target. To deploy the targets the following code is used in this project:

```
import random
from math import sqrt
import matplotlib.pyplot as plt
import numpy as np
gridsize=600
Range_sensor= 100
N_sensors=30
M_Targets=25
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets
for j in range (M_Targets):

    x_targets[j] = random.randint(0, gridsize)
    y_targets[j] = random.randint(0, gridsize)
    print("position of taget",j,"is",x_targets[j],y_targets[j])


    plt.scatter(x_targets[j],y_targets[j],100,'r','1')
    plt.text(x_targets[j] * (1 + 0.01), y_targets[j] * (1 + 0.01),
     j, fontsize=12)
plt.xlim(0,gridsize)
plt.ylim(0,gridsize)
plt.show()
```

Listing 4.2: Target deployment code

In Listing 4.2, the `M_Targets` represent the number of targets deployed. In this project, this number ranges from 20 to 100.

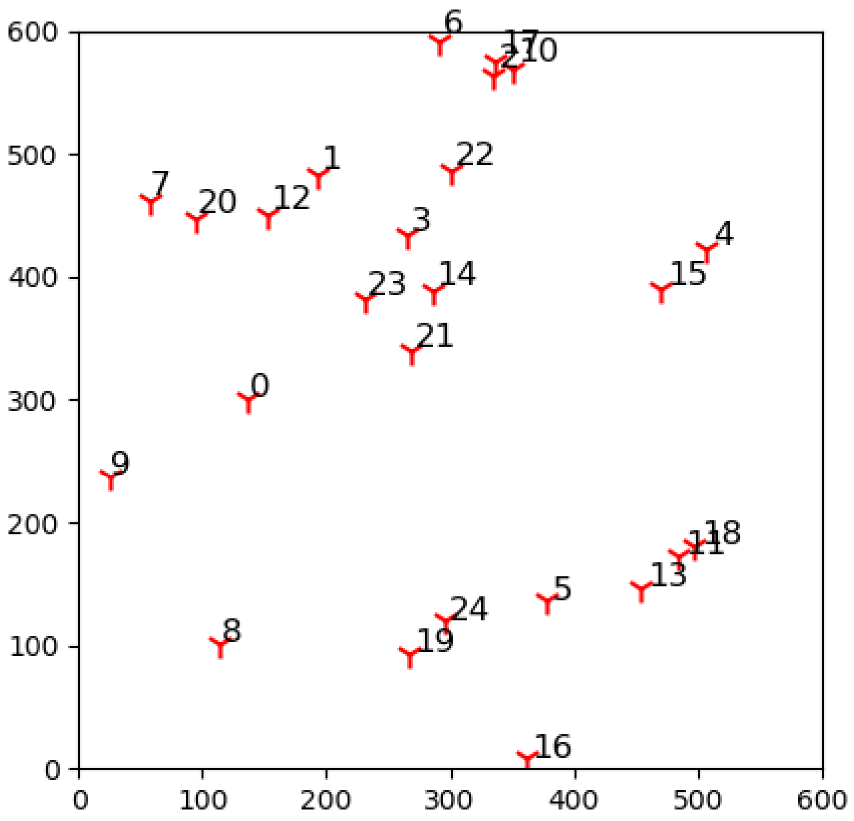


Figure 4.2: Deployed targets in a area of 600m×600m

In this figure 4.2 the red towers are the targets. The number of deployed targets are 25.

4.4 Sensor and Target Deployment

Figure 4.3 show the plot of sensor and target deployed in an area of the 600m×600m grid. Here, 10 sensors and 5 targets are deployed to draw this plot. And, the sensor is provided with a sensing range of 100m. The sensors are represented by green towers and targets are represented by red towers. Black circle around the sensor is their coverage area.

```python
 from numpy import random
import matplotlib.pyplot as plt
import numpy as np

gridsize = 600
Range_sensor = 100
N_sensors = 10
M_Targets = 5
x_sensor = [0] * N_sensors
y_sensor = [0] * N_sensors
x_targets = [0] * M_Targets
y_targets = [0] * M_Targets
max_time = 10
for i in range(N_sensors):
    x_sensor[i] = np.random.randint(0, gridsize)
    y_sensor[i] = np.random.randint(0, gridsize)
    print("sensor", i, "is at position", x_sensor[i],y_sensor[i])
    x1 = x_sensor[i]
    y1=y_sensor[i]
    plt.text(x1 * (1 + 0.01), y1 * (1 + 0.01), i, fontsize=12) #
    code for ploting the points with text

    plt.scatter(x1, y1,100,'g','1')
    circle = plt.Circle((x1, y1), Range_sensor, color='black',
    fill=False)
    fig = plt.gcf()
    fig.gca().add_artist(circle)

print("sensor", i, "is at position", x_sensor[i], y_sensor[i])
for j in range (M_Targets):
    x_targets[j]= np.random.randint(0,gridsize)
    y_targets[j]= np.random.randint(0,gridsize)
    x2 = x_targets[j]
    y2= y_targets[j]


    plt.scatter(x2,y2,100,'r','1')

    plt.text(x2 * (1 + 0.01), y2 * (1 + 0.01), j, fontsize=12)
    print("targets", j, "is at position ", x_targets[j], y_targets
    [j])
plt.legend()
plt.ylim(0,gridsize)
plt.xlim(0,gridsize)
# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')
# giving a title to my graph
plt.title('Sensors')
```

```

# show a legend on the plot
# plt.legend()

# function to show the plot
plt.show()
```

Listing 4.3: Sensor and target deployment code

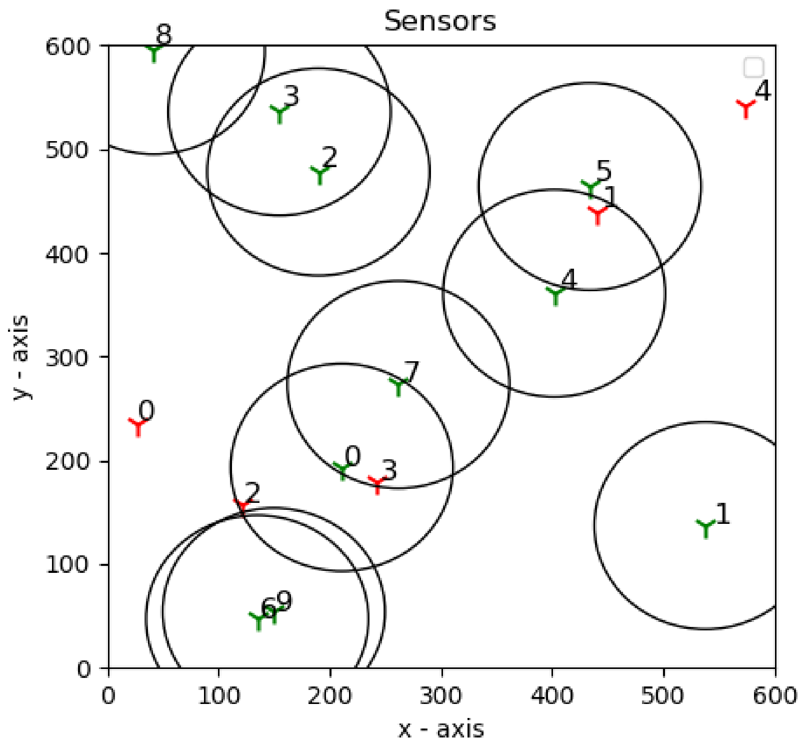


Figure 4.3: Sensor and target deployed in the same plane with sensor coverage area. Here, sensor denoted by green towers and targets by red towers. Target 1 is covered by sensor 5 and target 2 and 3 by senor 0

4.5 Algorithm Description

In this project, an adaptive learning algorithm is used. This algorithm helps in scheduling the sensor nodes action state. There is two action state of the sensor nodes. One is an active state, and another is sleep state. Inactive state sensor node communicates with its neighbors and covers the targets. Where is sleep mode sensor node do nothing? Also in case of energy usage, inactive state sensor node consume more energy while in sleep state no energy is consumed. At the initial state of the algorithm, every sensor is provided with an equal probability of covering the targets.

This algorithm is divided into three phase:

- **Initial Phase:**

  In this phase, all the sensor in the network is provided with learning automata. This helps the sensor node to select its state to active or sleep. In the initial stage, the probability of selecting the state is considered to be 0.5. Sensor nodes in the project are considered to be autonomous. Which means that they establish communication, transfer messages including their ID, position, and list of covered targets with their neighbor node autonomously. Such that they know which state to select at every condition. This phase is followed by learning and target monitoring phase.

Algorithm 1 Initial phase

```
  Input:
i Given, N number of sensors, M number of targets and R sensor
  sensing range
ii $a$ and $\epsilon$ learning automata parameter
  Output:
i Number of active sensors that cover the targets
  START
Initialize the Network parameters
Calculate the distance between sensor and target
Determine the coverage and covered targets in the network
if $(Rangeofsensor >= distancebetweensenorandtarget)$ then
    Set sensor mode to active and add them to list
    of active sensors by using learning algorithm
else
    Set sensor mode to sleep by learning algorithm
end if
iteration=iteration+1
while $AllTargetcovered$ do
end while
```

- **Learning phase**

Algorithm 2 Learning phase

```
for N number of sensor in network do
    for Each action of sensor node do
        initial probability = o.5
    end for
end for
for i=0 to iteration do
    for Every node in Network do
        Node= choose random action
        send action packet to neighbour nodes
        receive action packet from all neighbour nodes
        if Node action is Sleep and neighbour node could cover the target then
            Set the action mode of sender node to Sleep
            Decrease the probability vector to cover
            the target by using learning automata.
        else
            Increase the probability of sender node to cover the targets
        end if
        if Node action is Active and neighbour node could not cover the target then
            Then Set node mode to Active state
            Increase the probability vector of
             Learning automata to cover the targets
        else
             Go to sleep mode using learning automata
        end if
    end for
end for
```

In this phase, each sensor nodes are equipped with learning automata. At first, the node is selected randomly. Using purposed learning automata node select its state. Then it broadcast the message packet including its all information to its neighbors. By listening to this message all neighbor node reply to their state. Then the sender does the following actions:

- If selected Learning Automata state of sender node is active and Replying Neighbour node is also active:

  Then sender node changes its state to sleep by decreasing its probability vector value for covering the target.

- If selected Learning Automata state of sender node is active and Replying Neighbour node is also sleep:

  Then the sender node will be in active state, and it will cover the target in the network. And the probability vector of covering target will increase.

This phase continuous till all the target of the network are covered. In every round of operation, the active state sensor node probability is increased.

- **Target monitoring phase:**

Algorithm 3 Target Monitoring phase

```
for N number of sensor in network do
    Select Best action()
    if Best action()=Active then
        Node state = Active
        Current Coverage set= Current Coverage U Node
    else
        Node State = Sleep
    end if
end for
Monitor the target till end of target monitoring phase
```

To start monitoring the target, each node selects their state according to their probability vector of learning automata. Then the node with the active state will begin tracking the target. When all target is monitored in the network, then this phase ends with the starting of the initial stage and learning phase.

Chapter 5

Implementation And Result Analysis

This chapter discusses the different implementation steps, results obtained, challenges and the different error sources that could have affected the experiments.

Also here proposed algorithm is evaluated, which is known as Adaptive Learning Automata Algorithm (ALAA). The evaluation is done by conducting some computer simulation of the network with sensors and targets deployed randomly in the space of 600m$\times$600m area.

For all the experiments following parameter are used:

- N, the number of sensors deployed randomly. Its value ranges between 10 and 80.
- M, the number of randomly deployed targets. Its value ranges between 4 and 60.,
- R, the sensing range of the sensor. Its value ranges between 50m and 600m.
- Experiment is done 1000 times to get a result.
- There is a use of random seed in the script. This helps in using the same configuration for doing the experiments.
- Learning parameter
  - Lambda(λ), its value ranges between 0 .000 1 and 0.4
  - Epsilon(ϵ), its value remains constant, which is 0.01 for all experiment.

Among the parameters as mentioned earlier, only the parameters that need to be tested are changed for the experiment, and other remains constant.

5.1 Experiment One

This experiment is done to find the impact of increasing sensing range of the sensor to get minimum active sensors. Here, the main goal is to obtain as minimum as possible active sensors that cover all the targets in the network. The experimental setup with result is illustrated by Table 5.1 and Table 5.3. These results are projected in Figure 5.1. This Figure 5.1, shows for 20 sensors and 10 targets with increasing value of sensing range. There is no effect of the number of targets deployed for obtaining minimum active senors. This is because the target are uniform randomly deployed in the network. And, even 15 target and 20 sensors are taken for experiment, there is a notable change in the results for getting minimum active sensors.

Table 5.1 Environment setup and the result obtained by varying sensing range of sensors with the use of learning automata with 20 sensors and 10 targets

Range of Sensor (R)	Average Minimum Active Sensor
150	4.532
200	3.356
250	2.599
300	2.112
350	1.387
400	1.179
450	1.123
500	1.112
550	1.027
600	1.003

Table 5.2 Environment setup and the result obtained by taking small values of the sensing range of sensors with the use of learning automata with 20 sensors and 10 targets

Range of Sensor (R)	Average Minimum Active Sensor
50	12.467
60	8.31
70	6.631
80	6.482
90	6.256
100	5.726

And, the Table 5.2 shows the impact of the smaller value of sensing range to find minimum active sensors. From the results, one can observe that there is the same pattern as the result as obtained by taking a more significant value of sensing range.

So, a conclusion can be made that with an increasing range of sensor there is a need for less number of the active sensor to cover the targets. The reason is that with larger sensing range sensors can cover more targets. This means that With the use of a less active sensor there is a use of less energy to monitor the target and thus maximizing the lifetime of the deployed sensor network.

Figure 5.2 shows the simulation result. The green towers represent deployed sensors, and red towers represent deployed targets. The black circle with a number represents the sensors that are in active state and covering the target. And, sensors that are in sleep state represented by yellow dotted coverage.

Table 5.3 This table show results obtained while taking 15 targets, 20 sensors and by increasing the sensor sensing range from 150m to 600m

Range of Sensor (R)	Average Minimum Active Sensor
150	5.958
200	3.788
250	2.552
300	2.125
350	1.625
400	1.198
450	1.122
500	1.042
550	1.040
600	1.02

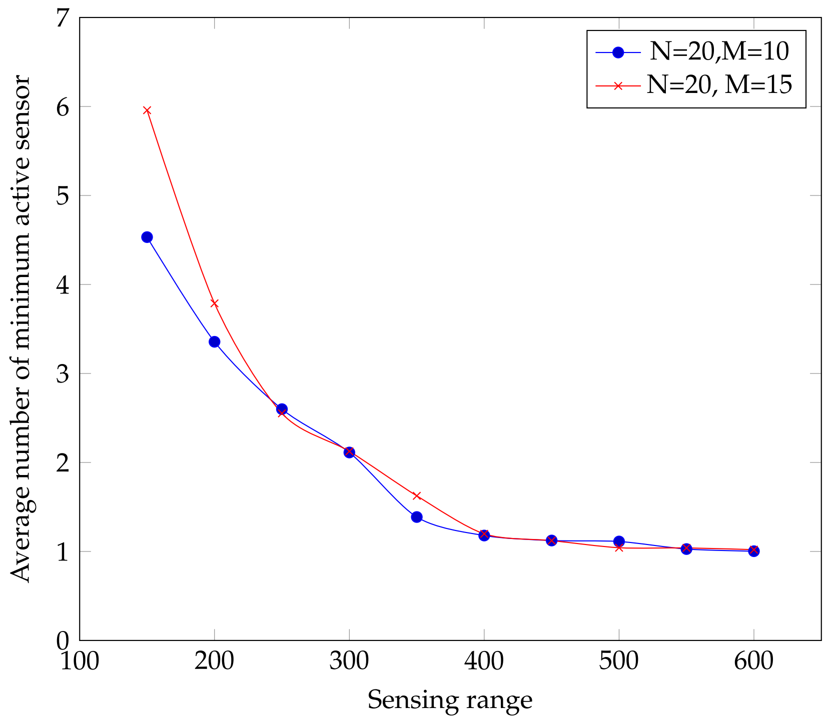


Figure 5.1: Blue line graph plot show results obtained by changing sensing range of sensor from 150m to 600m with a step of 50m and there are 10 targets, 20 sensors deployed and red line graph plot show results obtained by changing sensing range from 150m to 600m with a step of 50m and by deployment of 15 targets and 20 sensors

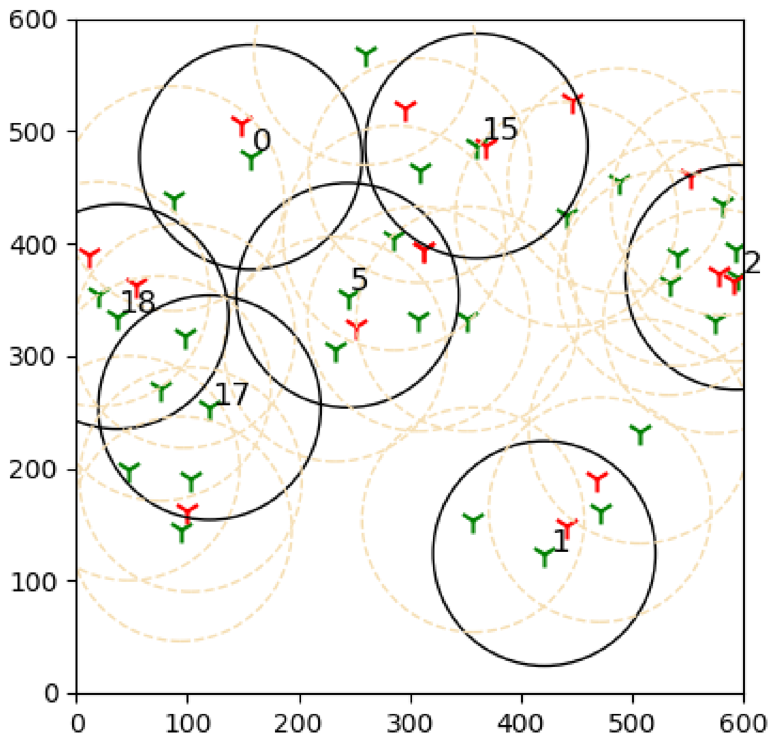


Figure 5.2: One of the final simulation results of the experiment where 7 active sensors are able to cover all the 15 targets. The sensing range is 200m.

5.2 Experiment Two

Table 5.4 This table show results obtained while taking 10 targets and sensing range 300m with an increasing number of the sensor from 20 to 30

Number of sensor(N)	Average Minimum Active Sensor
20	2.136
21	2.132
22	2.130
23	2.125
24	2.109
25	1.879
26	1.768
27	1.520
28	1.280
29	1.113
30	1.023

The goal of this experiment is to investigate the relationship between a number of the sensor on obtaining minimum active sensors in the network. For this number of the sensor is taken in between 20 and 30 with a step of 1. The results obtained is included in Table 5.4, Table 5.6 and Table 5.5. The first two table include result obtained by taking 10 and 15 targets with a sensing range of 300m and 400m. And, next table includes 15 targets with a 100m sensing range of a sensor. The obtained results are shown by Figure 5.3 which shows that with an increase in the number of sensors leads to obtaining the minimum number of active sensors. Also, with increasing the number of targets and sensing range, there is little effect on the network. So, both curve is noticeably equal. This shows that with the increase in the number of sensors there is a decrease in need of more active sensors. Thus, this helps in maximizing the network lifetime.

The Table 5.5 represent the results obtained by use of small sensing range. Figure 5.4 shows that with small sensing range there is need of more sensors to cover the targets. But increase in the sensing range, results in a decrease in the number of active sensors that are used for covering the target which shows the same trend of results as it has been taken for larger values of sensing range shown by Figure 5.3.

So, from this experiment one can see that how the increasing number of sensor effect on obtaining minimum active sensors. Also, there is to notice that results are in the same trend even there is selection of large or small sensing range.

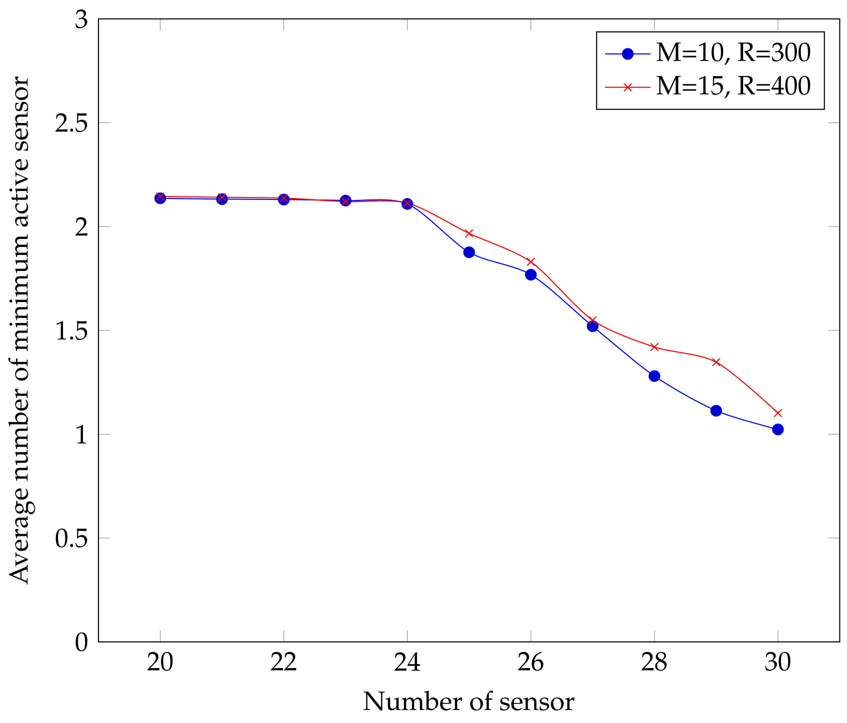


Figure 5.3: Plot showing impact of increasing number of sensors on obtaining the average number of minimum active senors in the experiment

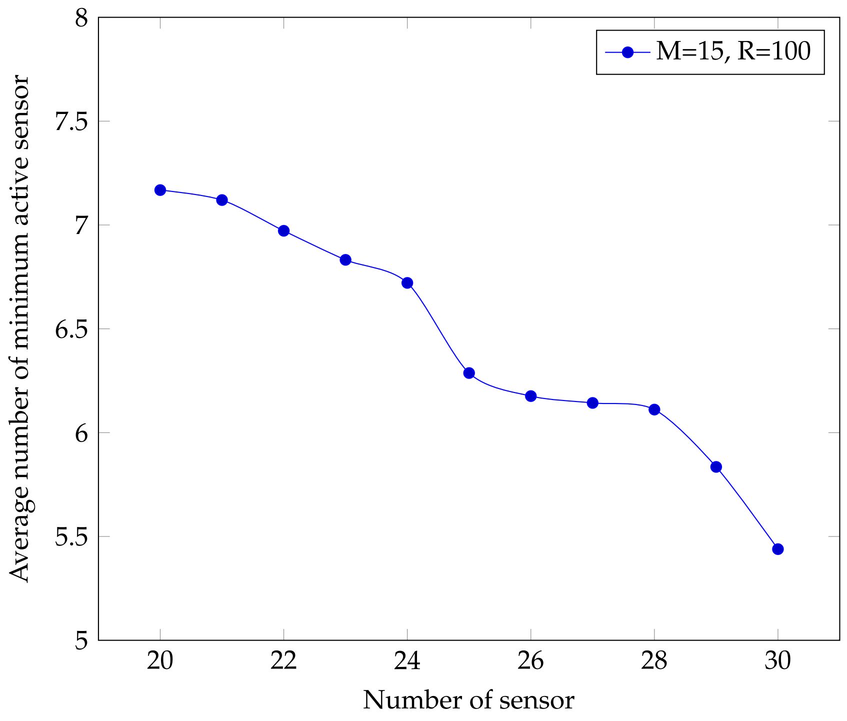


Figure 5.4: Plot showing the impact of increasing number of sensors by taking 15 targets and small sensing range of 100m to obtain average minimum active senors in the experiment

Table 5.5 Tabulation of results obtained while taking 15 targets and sensing range 100m with the increasing number of the sensor from 20 to 30

Number of sensor(N)	Average Minimum Active Sensor
20	7.168
21	7.120
22	6.972
23	6.832
24	6.721
25	6.287
26	6.176
27	6.143
28	6.111
29	5.835
30	5.439

Table 5.6 This table show results obtained while taking 15 targets and sensing range 400m with the increasing number of the sensor from 20 to 30

Number of Sensor (N)	Average Minimum Active Sensor
20	2.145
21	2.140
22	2.135
23	2.120
24	2.110
25	1.967
26	1.830
27	1.549
28	1.420
29	1.347
30	1.102

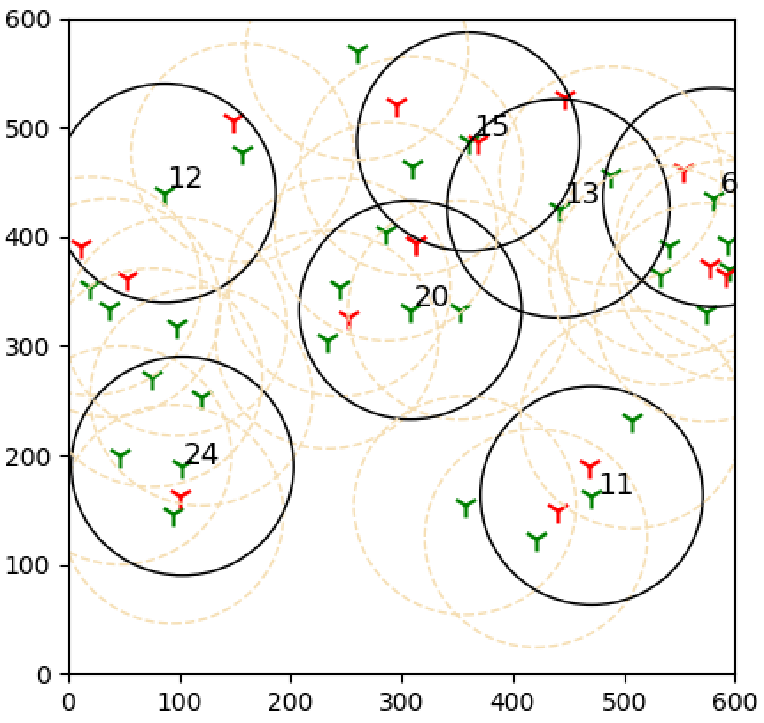


Figure 5.5: Simulation result of the experiment showing the impact of the density of sensors in a network

5.3 Experiment Three

This experiment is conducted to investigate the impact of the density of target in obtaining average minimum active sensors in a network. To do this, the target is taken between 4 and 26 with 30 sensors and sensing range of 100m. Figure 5.6

Table 5.7 This table show results obtained while taking targets between 4 and 26 with 30 sensors and sensing range of 100m

Number of Target (M)	Average Minimum Active Sensor
5	4.861
10	7.549
15	8.33
20	14.345
25	18.795

Table 5.8 This table show results obtained while taking targets between 4 and 26 with 30 sensors and sensing range of 300m

Number of Targets (M)	Average Minimum Active Sensor
5	2.261
10	2.477
15	2.792
20	3.055
25	3.099

shows the result obtained after experimenting. From this result, one can see that there is an inverse relationship between a number of target and average minimum active sensors. This means that when there is an increase in the number of targets than the more number of active sensor needed to cover them. But, there is the effect of sensing range with the increasing number of targets.

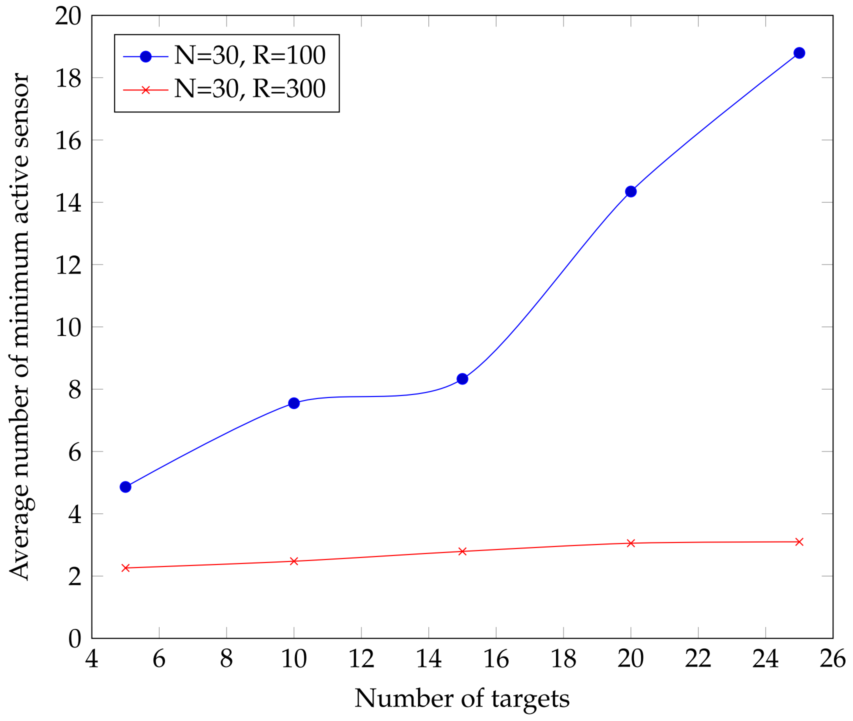


Figure 5.6: Plot showing impact of increasing number of targets on obtaining average minimum active senors in the experiment

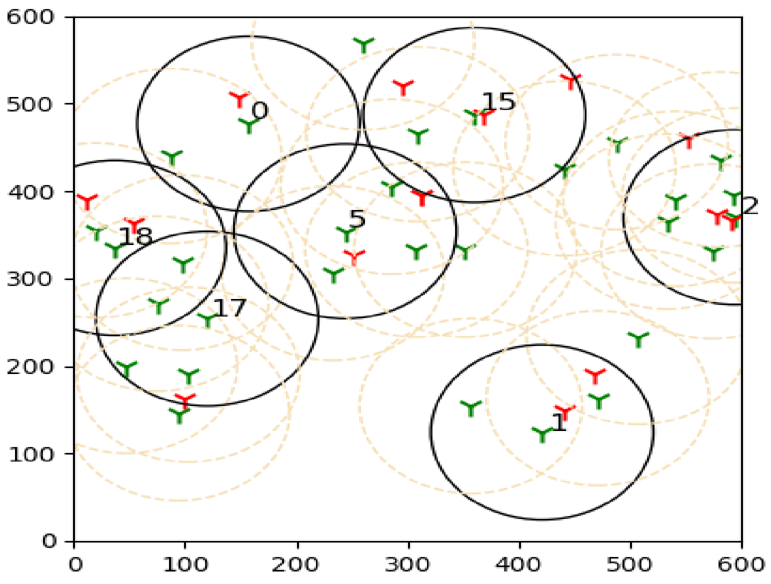


Figure 5.7: Simulation result on impact of target density on obtaining average minimum active sensors

5.4 Experiment Four

The primary goal of this experiment is to investigate the impact of sensing range and sensor density in a vast network. An extensive system is defined as the network where a large number of sensors and targets are deployed. At first, the experiment is done by varying the range of sensor between 150m and 600m with 70 sensors and 50 targets. Then, the next test is done by increasing the density of sensors to evaluate the algorithm performance to obtain minimum active sensors. The Table 5.9 and Table 5.10 shows the experimental setup and the results achieved.

Table 5.9 This table show results obtained while taking 50 targets and 70 sensors by increasing the sensor sensing range from 150m to 600m

Range of Sensor (R)	Average Minimum Active Sensor
150	11.752
200	8.869
250	7.557
300	7.054
350	6.694
400	6.523
450	6.512
500	6.510
550	6.465
600	6.241

Table 5.10 This table show results obtained while Increasing sensor number from 70 to 80 with 50 targets and sensing range 300m

Number of Sensor (N)	Average Minimum Active Sensor
70	2.145
71	2.140
72	2.135
73	2.120
74	2.110
75	1.967
76	1.830
77	1.549
78	1.420
79	1.347
80	1.102

The figure 5.8 shows the plot of the results from the Table 5.9. From the plot, one can see that there is constant declination in the plot with an increase in sensing range. This provides the effect of increasing sensing range of the sensor in a large network. This is because With increasing sensing range the coverage area of sensor

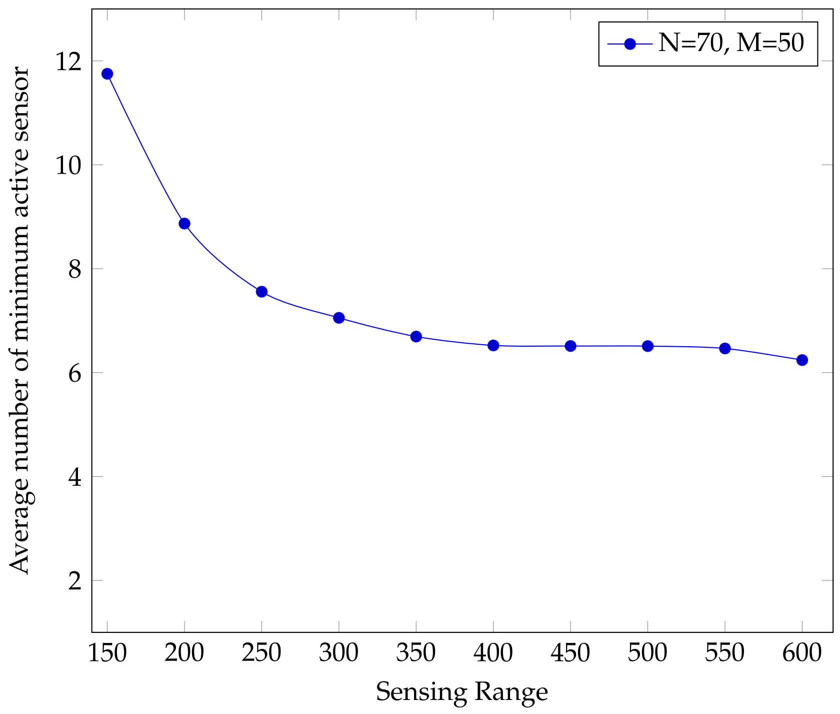


Figure 5.8: An extensive network consisting of 70 sensors and 50 targets with a change of sensing range from 150m to 600 m

increases. Larger the coverage area, there is a high probability that more target will fall inside this range. Such that it can monitor the number of targets.

So, there is less number of sensor required to be in the active state for monitoring the target. All other sensors that are not in the active state can go to the sleep state to save their battery. And they can be used in the next round of experiment.

From above results, one can conclude that with an increase in the sensing range of sensor there is a decrease in the number of active sensors in the large network. Figure 5.9, show plot of data from Table 5.10. It indicates that when more sensors are deployed in a large network. Then there is more probability of getting less active sensors to monitor the targets. To obtain this results one should decrease the value of the probability parameter corresponding to an increase in the number of sensors.

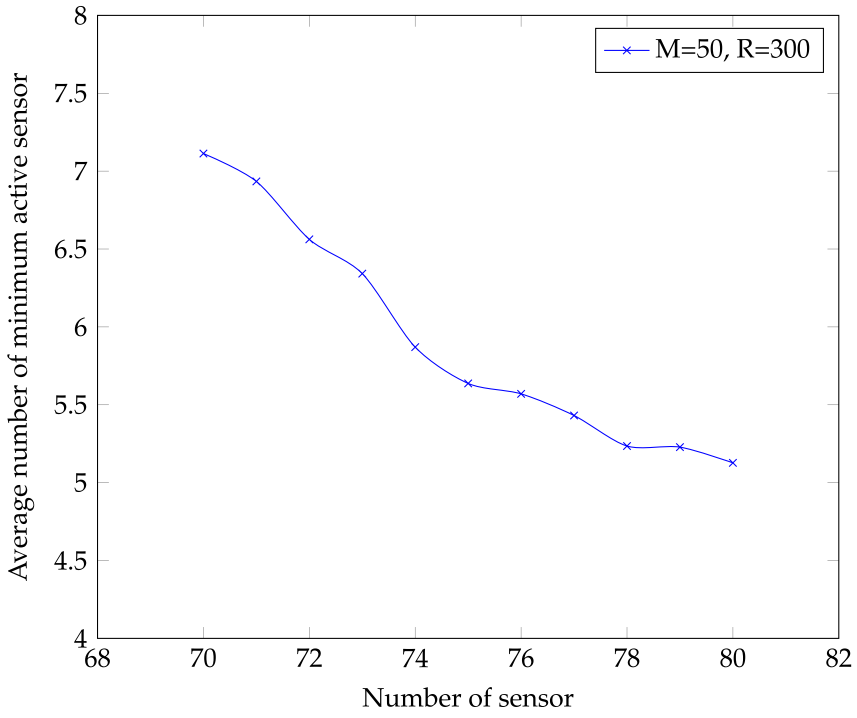


Figure 5.9: The result of a large network consisting of 50 targets and sensing range of 300m with the increasing number of sensors from 70 to 80

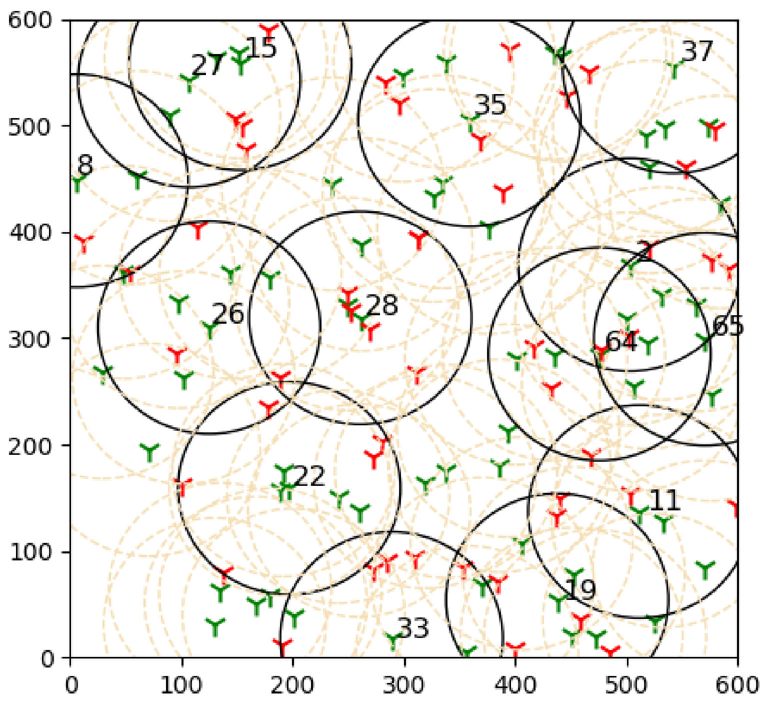


Figure 5.10: Simulation result showing the impact of sensor density and sensing range on obtaining average minimum active sensors in the large network

5.5 Experiment Five

The main goal of this experiment is to examine the relationship between the learning parameter rate with minimum active sensors. In the proposed algorithm, there are two parameters used. One is lambda (λ), and another is epsilon (ϵ). The lambda "λ" parameter of learning automata is varied to get the results.

To do this experiment, sensors between 40 and 80 are taken with 30 targets. Every sensor is provided with a sensing range of 400m. Then, by taking the different value of learning parameter lambda, "λ" experiment is formulated. Here, the value of lambda range from
"λ= 0.1", "λ=0.01", "λ= 0.001" and "λ=0.0001". The results are tabulated in Table 5.11.

Table 5.11 Observation table showing result of experiment after taking the value of learning parameter lambda "λ" as "λ= 0.1", "λ= 0.01" ,"λ= 0.001" and "λ= 0.0001" with sensors between 40 and 80 including 30 targets and sensing range is 400m

Lambda "λ" Number of sensors	**λ= 0.1 Average Number of active sensors**	**λ= 0.01 Average Number of active sensors**	**λ= 0.001 Average Number of active sensors**	**λ= 0.0001 Average Number of active sensors**
40	9.844	2.912	1.735	1.345 ☑
50	13.189	3.726	1.851	1.436 ☑
60	16.926	5.082	1.967	1.483 ☑
70	20.599	6.635	2.176	1.508☑
80	24.049	7.576	2.078	1.526 ☑

In Table 5.11 one can see the results obtained after experimenting. From this observation, one can explain that there is a decrease in the requirement of an average number of minimum active sensors while decreasing the value of the learning parameter. But, with a reduction in the value of the learning parameter, there is an increase in the number of iteration which gives more precise results. This phenomenon leads to an increase in program execution time and high memory usage. Another explanation is that if there is an increase in the number of sensors, then there should be a decrease in learning parameter value to get the best results.

The results of this experiment are shown in Figure 5.11, which shows a bar-graph plot between the learning parameter versus the number of sensors. The observation provides that if the number of sensors is increased in the deployed environment, then a small value of learning parameter has to be taken to get good results. This gives use of less active sensors to monitor all the targets.

This result provides an idea of obtaining good scheduling capacity of our proposed algorithm. In another word, this decreasing value of learning parameter value helps the proposed algorithm to schedule the sensor nodes activity more efficiently. Which results, to use of minimum sensors and thus maximizes the network lifetime. Also, it shows that for an extensive network with a large number of sensors, there should be variation in the value of learning parameter to get the best results.

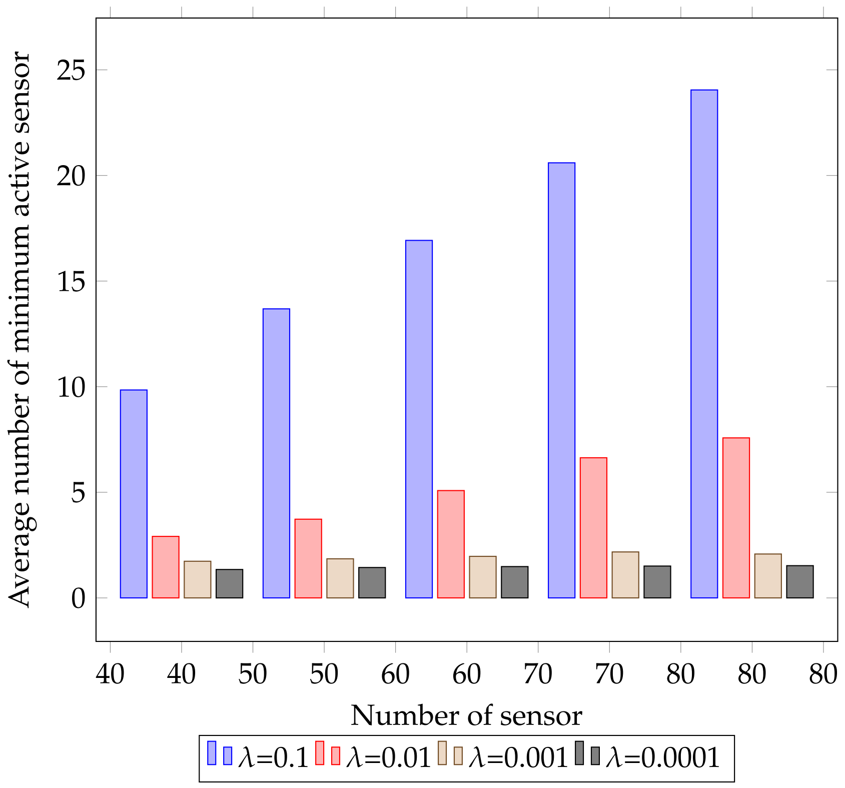
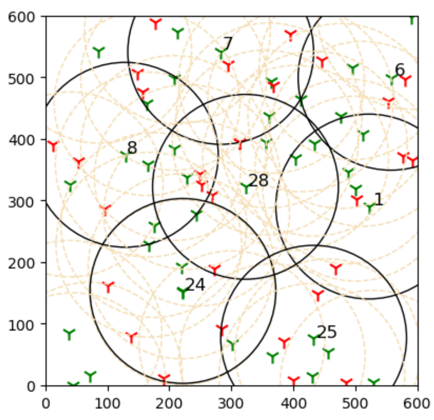
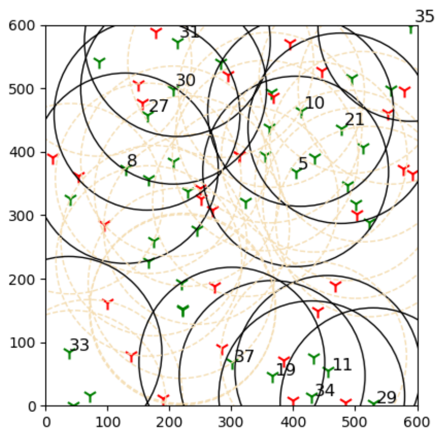


Figure 5.11: Bar plot of the data obtained in Table 5.11 showing the effect of decreasing value of learning parameter values

So, the conclusion that is obtained from this result is that with a small amount of learning parameter the proposed algorithm provides proper scheduling to increase the network lifetime with an increase in the sensors network.



(a) Result of taking small values
of learning parameter



(b) Result of taking larger values
of learning parameter

Figure 5.12: Simulation results of the experiment to investigate the impact of the learning parameter of the Proposed learning automata algorithm.

5.6 Experiment Six

The goal of this experiment is to check how correct are our results produced by our proposed learning automata algorithm. For this, the experimental set up is done by taking 9 targets and number of sensors ranging from 10 to 20. The sensing range of all sensors is constant, and its value is taken 100m. Then, the experiment starts by first finding the number of average minimum active sensors by using the proposed learning automata algorithm. And, next with the same configuration brute force testing is done. This brute force method produces 2^n combination of the deployed number of sensors. Here, "n" is the number of sensors used in the experiment. From this combination, it results in only the combination of best active sensors. Results from both methods are observed and compared to find the best results.

Table 5.12 Table showing experimental data set and the results obtained after experimenting with the proposed learning automata algorithm(ALAA) and brute force (2^n) method. Here, the number of sensors deployed range from 10 to 20 with a step of 2 and sensing range parameter value is 100m with the used number of targets is 9.

Number of Sensor (N)	**Average Minimum Active Sensor (ALAA)**	**Number of Minimum Active Sensor (Brute force(2^n))**
10	6.450	5
12	5.671	5
14	5.197	5
16	5.066	5
18	5.076	5
20	5.155	5

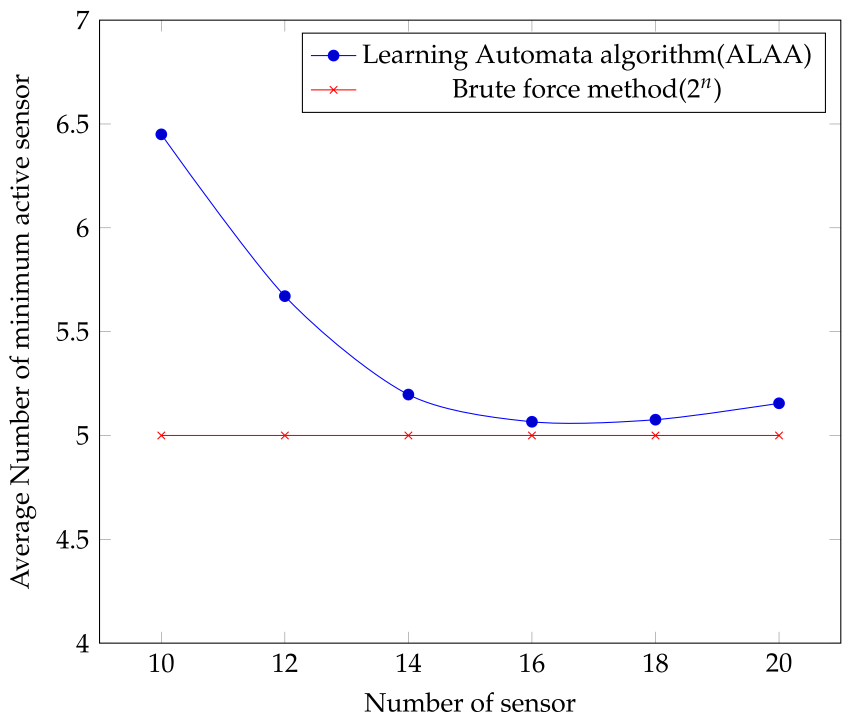
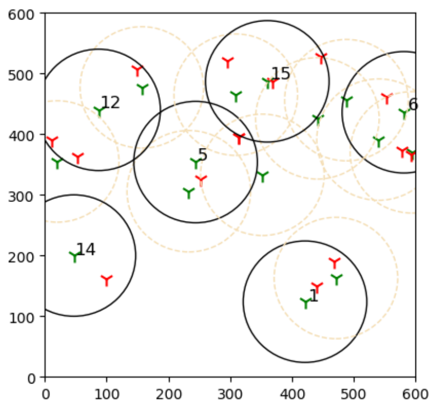


Figure 5.13: Plot showing graph of results from 2^n method and brute force method

The Table5.12 provides the results obtained by formulating the experiment. This result is plotted in Figure 5.13. From this, one can see that the results obtained by the use of brute force method(2^n) and proposed learning automata algorithm(ALAA) method. Brute force method gives the linear curve of results. But, the results from the proposed learning algorithm first inclines to some extent, and after that, it shows linear behavior. This provides the proof that brute force gives optimal results than the proposed algorithm. And, thus it helps in finding


(a) Result of proposed algorithm(ALAA)

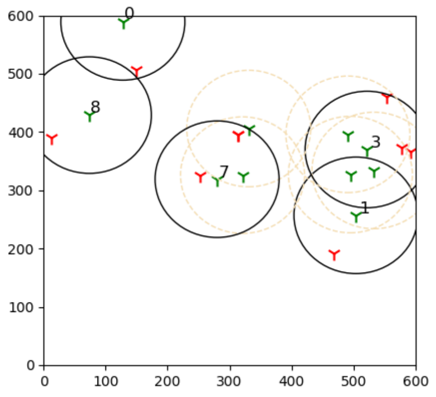

(b) Result of brute force method (2^n)

Figure 5.14: Simulation result obtained after formulation of the experiment using the Proposed learning automata algorithm and brute force method

the efficiency of our proposed algorithm to meet our goal.

Figure 5.14 shows the results of this experiment. In the figure, green towers are the sensors, and red towers are the targets. The sensors that are covering the targets are denoted by the number, and black circles denote their coverage. This result concludes that brute force gives an optimal solution than by the use of proposed learning automata algorithm.

5.7 Experiment Seven

This experiment is performed to find the efficiency of the proposed learning algorithm by comparing it with other algorithms. For this, LADSC scheduling algorithm from Paper [7] is taken for comparing results of proposed learning automata algorithm. These algorithms are compared first by using the density of sensors and then by using a different range of sensors. The algorithms are evaluated from their property of using the minimum number of best active sensors and covering the targets. For the first comparison, there is a use of 9 targets with sensors between 9 and 25 having a sensing range of 100m. And, the second comparison is made with 15 targets, sensing range of sensor increase from 50m to 100m and the number of the sensor is 20.

Table 5.13 shows the number of active sensors under each algorithm as mentioned in the table with their covered targets. Here, the effect of the density of sensors to obtain an average number of minimum active sensors is evaluated by using these algorithms for this sensor number increased from 10 to 24 with a sensing range of 100m. The results obtained are projected in Figure5.15. From these obtained results, it can be observed that our proposed scheduling algorithm provides better results in comparison to LADSC algorithm for obtaining less number of active sensors to monitor all the targets. Even in some case, the LADSC algorithm gives the effect of using less active sensors, but it is unable to cover all the target. So, it is less efficient than our algorithm. This provides the fact that our

Table 5.13 Comparing proposed algorithm with LADSC algorithm mentioned in paper [7] and checking the correctness of the results by use of brute force method with 9 targets and sensors between 9 and 25 with a sensing range of 100m

Number of Sensor(N)	ALAA		LDSC		Brute Force	
	Active sensors	**Target covered**	**Active sensors**	**Target Covered**	**Active sensors**	**Target covered**
10	6.0	9.0	7.0	9.0	6	9
12	5.0	9.0	7.0	9.0	5	9
14	5.0	9.0	6.0	8.0	5	9
16	5.0	9.0	6.0	9.0	5	9
18	5.0	9.0	4.0	7.0	5	9
20	5.0	9.0	6.0	9.0	4	9
22	4.0	9.0	5.0	9.0	4	9
24	4.0	9.0	5.0	9.0	4	9

algorithm uses less amount of energy than LADSC algorithm and so it helps in maximizing the lifetime of a sensor network. And, produce the optimized results.

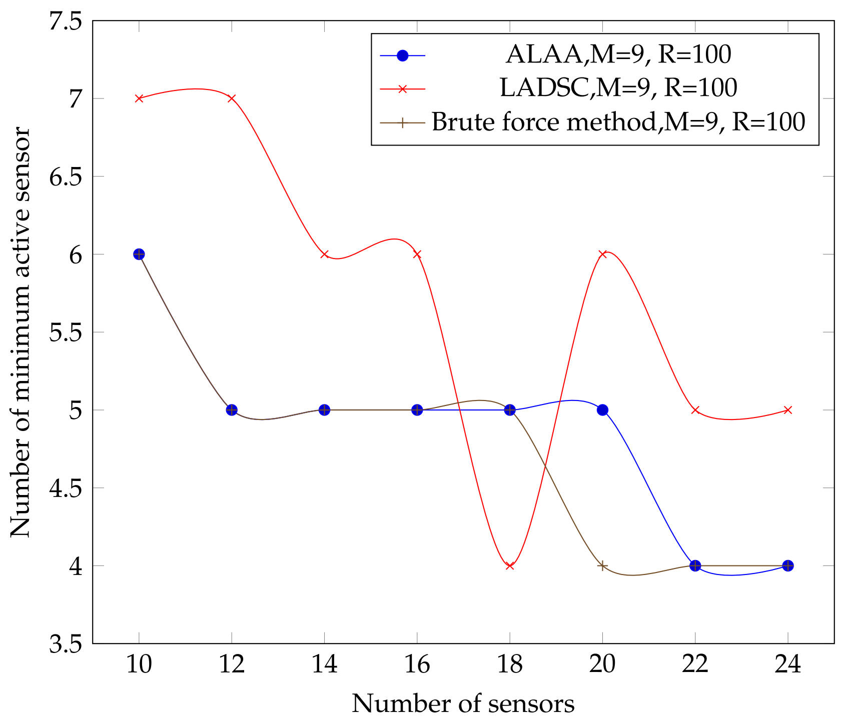


Figure 5.15: The plot of results obtained by Comparing proposed Adaptive Learning Automata Algorithm (ALAA) with Learning Automata Disjoint coverage set (LADSC) Algorithm and Result checking by brute force method. Here 9 targets are used with the sensor between 9 to 25 with a sensing range of 100m.

Table 5.14 Comparing the proposed algorithm with LADSC algorithm mentioned in paper [7] and checking optimization result by using brute force method with the deployment of 10 targets and 15 sensors with varying sensing range from 50m to 100m

Sensing Range(R)	ALAA		LDSC		Brute Force	
	Active sensors	**Target covered**	**Active sensors**	**Target Covered**	**Active sensors**	**Target covered**
50	9.0	10.0	10.0	9.0	9	10
60	9.0	10.0	11.0	10.0	9	10
70	8.0	10.0	10.0	9.0	8	10
80	8.0	10.0	9.0	10.0	8	10
90	8.0	10.0	7.0	8.0	7	10
100	7.0	10.0	8.0	10.0	7	10

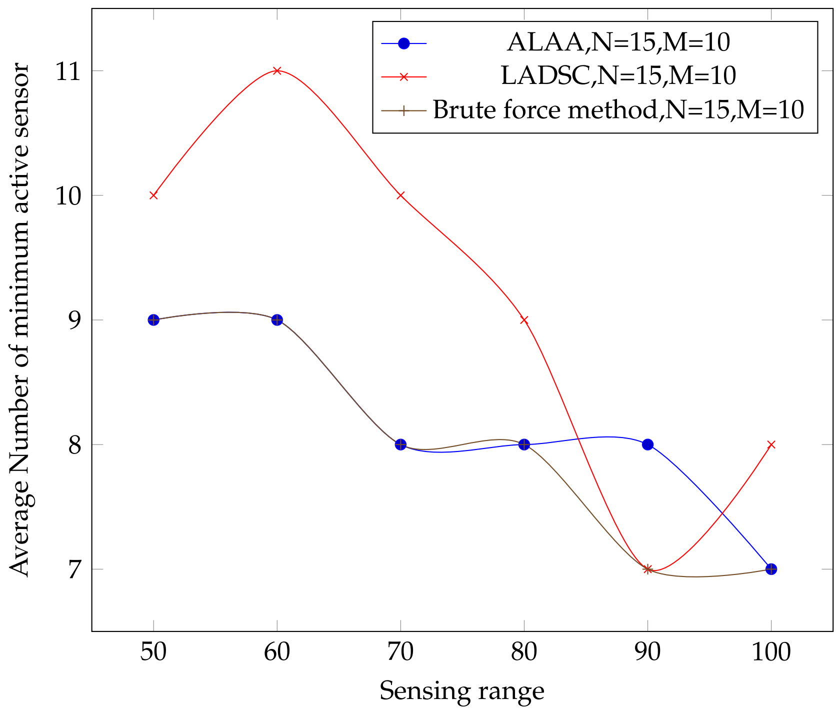
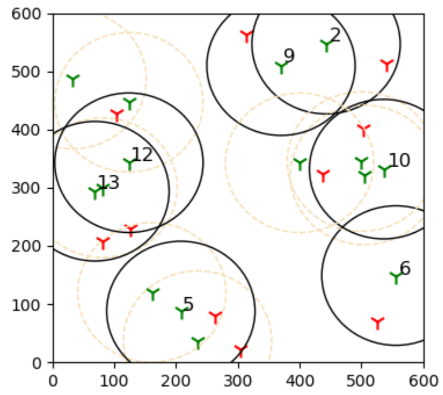
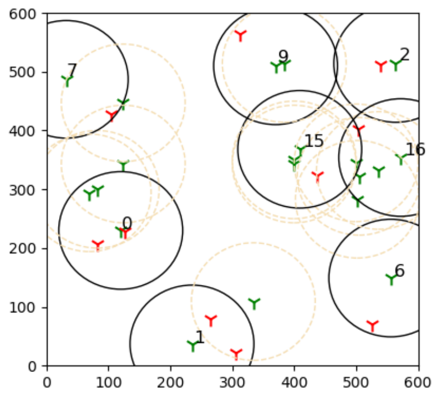


Figure 5.16: The plot of results obtained by Comparing proposed Adaptive Learning Automata Algorithm (ALAA) with Learning Automata Disjoint coverage set (LADSC) Algorithm and checking results by use of brute force method. Here 15 sensors and 10 targets are used with varying sensing range between 45m and 110m

Figure 5.16 shows the plot of the results obtained after formulating two algorithms. From this figure, one can easily see the different pattern of the results. In between this two algorithm, the proposed algorithm gives better results. Even there at some point in the graph, the LADSC algorithm shows that they use less active sensors. But in this condition they are not covering all the targets. But



(a) Result of proposed algorithm(ALAA)



(b) Result of LADSC

Figure 5.17: Simulation result obtained after formulation of the experiment using Proposed learning automata algorithm and LADSC algorithm with a varying sensing range of sensors from 50m to 100m

our proposed algorithm is covering all the targets. So, it proofs that the proposed algorithm is more consistence than LADSC algorithm.

Figure 5.17 shows that there is need of more sensors to cover all the target when there is use of LADSC algorithm but proposed algorithm use less number of active sensors. Thus, this result concludes that the proposed algorithm is capable of obtaining optimized results.

Chapter 6

Discussion

This chapter discusses how the project is evolved and implemented. Along with this, there is a discussion about the results obtained to find if this result meets our project goal or not. And, if not then what errors occur to do so are also highlighted in this chapter.

6.1 Problem statement

To start the project, there should be a goal. The problem statement of the project provides this goal. Within the boundary of this goal, the project is done to get the results. Our project problem statement is mentioned in section 1.4, the main goal of this statement is to find energy efficient sensor network using learning automata algorithm. The efficiency is measured by formulating different experiments. To obtain the goal of the problem statement, there is a use of different parameters in the network.

- N = Number of sensors
- M = Number of targets
- R = Sensing range
- LA = Learning Automata
- λ, ϵ = Learning Parameter

The results are obtained by varying density of sensor nodes and targets, changing sensing range, increasing and decreasing learning parameter value and comparing proposed algorithm results with others algorithm described in related papers.

6.2 Algorithm and Experiment analysis

In this project, an adaptive learning automata algorithm is used to schedule the sensor nodes. This algorithm provides sensor node for selecting their appropriate state to cover the deployed targets. This algorithm is developed and deployed in the python script. The main goal of the algorithm is to find a minimum number of active sensors to cover all targets with maximizing the lifetime of the sensor network as a whole. And, by using the sensing range and an increasing number of sensors, it is compared with LADSC algorithm from paper [7] to show the efficiency of our proposed algorithm.

The following is a brief discussion of experiments and result obtained by formulating each of the experiments of the project.

6.2.1 Experiment One

This experiment is used to see the impact of increasing sensing range of the sensors in the network. From the results obtained in Table 5.1 and Table 5.3, one can see that there is the decreasing pattern of the average number of minimum active sensors. This means that there is less number of active sensors in the network with other being in sleep mode and saving battery power for later use. So, it justifies our stated goal of the project.

6.2.2 Experiment Two

In this experiment impact the sensor, density is tested. From the results obtained in Table5.4 and Table5.6, one can see that there is a decrease in the number of active sensors with the increase in the density of sensors.Thus, satisfies the goal of the project.

6.2.3 Experiment Three

In this experiment impact of targets deployed in the network for obtaining minimum active sensor is investigated. From the results, the conclusion can be made that there is an inverse relationship between the target and obtaining of minimum active sensors.

6.2.4 Experiment Four

This experiment highlights the impact of the proposed algorithm in a larger network. This is done both by increasing the number of sensing range as well as a number of the sensor. From the results in 5.9 and 5.10, it proofs that even in large networks there is the probability of obtaining energy efficient network is high.

6.2.5 Experiment Five

This experiment is done to check the impact of learning parameter value on the algorithm performance to get desired results. From the results obtained in Table5.11 and Table5.12, one can say that to get precise results the value of the learning parameter must be taken as smaller as possible with the increasing number of a sensor.

6.2.6 Experiment Six

This experiment glance on the correctness of the results. To check this, there is the use of brute force (2^n) method. The result shows that proposed algorithm produce the correct results.

6.2.7 Experiment Seven

This experiment is formulated to check the efficiency of a proposed algorithm. This is done by comparing the proposed algorithm with preceding algorithms that are described in the related work section of the thesis. The results obtained show that proposed algorithm outperforms similar algorithms in case of prolonging network lifetime.

6.3 Project

There are many issues of the wireless sensor network to do research on. This project is evolved to solve the issue of target coverage with energy conservation in wireless sensor networks. Due to the use of low powered batteries in sensors there is chance of energy issue. This issue effect in the lifetime of the sensor network as a whole. For one who wants to know more about the energy issue in wireless sensor networks can take a glance on the related work section of this thesis. This section includes different research and methodology that are used to solve this problem. Also, there is a description of how different algorithms are developed and used to solve the issue of target monitoring to conserve energy.

From research paper findings, this thesis is formulated with the development and design of infrastructure and learning automata algorithm to schedule the sensor nodes to monitor the targets.

The proposed algorithm is designed to give as minimum as possible active sensors to cover all targets. Such that there is the use of less number of sensor nodes which need the use of less power. Which in other hand helps in preserving some instance of the energy of the whole network. The consistency of the proposed algorithm is evaluated by doing experiments and simulations. These experiments are done by changing the values of a parameter of the proposed algorithm. This is done to see the impact of these parameters in finding appropriate results. The experimental results populated in table and graph in implementation and result analysis section chapter 5 of this thesis provide proof to the consistency of the proposed algorithm.

6.4 Error sources and challenges

This section discusses the errors that are encountered during the formulation of experiments and trials taken to tackle this errors to reach the project goal. During formulating this project there comes a lot of technical and logical problems. Some problems are solved easily, and some take time to analyze them and find the solution. These errors occur due to some misleading experimental setup and scripts. With the help of my supervisor and the Google search engine, the solution to the problem is found, and error is eliminated.

The first error is encountered while experimenting with the large network while selecting the more significant value of learning parameter of the proposed algorithm.

In this, when there is an increase in the network size, then there is a need for a decrease in the value of the learning parameter. This is because larger network converges fast with a smaller amount of learning parameter. Such that the results that obtained may not be the exact solution that it should give. For the answer to this problem there is the use of a smaller value of learning parameter in case of large networks.

Second, is due to time limitation to complete the project. The target monitoring phase by energy utilization is not formulated here. So, by obtaining the minimum number of sensors, it is concluded that a minimum number of sensor utilize less power to cover the targets and thus rest of the sensor hold their energy for later use. And, thus the network lifetime is maximized.

6.4.1 Inconsistent results generation

Most of the results obtained in this project are consistency. But there is one case where the project has to deal with inconsistent results. This case is of while selecting the value of the learning parameter to implement the proposed algorithm impact on the large network. When the experiment is started then if the selection of learning parameter value is too large or too small that will not fit the experiment then it will produce inconsistent results. For example, for the increased density of sensor, there will be need of decreasing value of learning parameter. So, it will give consistent results. Otherwise, our algorithm will give strange results.

Also, there is a problem on taking the smaller value of the learning parameter because it takes them a too long time to get converge even though it generates precise results.

Chapter 7

Further work

This chapter includes the suggestions for our current work. Furthermore, improvements of the developed algorithms and models are suggested as future work as an extension of this study. In this Chapter, suggestions to our current work and list new features that can be implemented are included.

7.1 Improvements

This project deals with the concept of maximizing the lifetime of the wireless sensor network by minimizing the use of sensors. In this project, there is the use of Adaptive learning automata algorithm to formulate the experiments. Results are obtained by doing many experiments. For each experiment, different parameters value as per need is varied by keeping other parameters constant. This project is capable of showing how efficient can be the network lifetime of wireless sensor networks if there is consideration in using of minimum active sensors.

To get more precise results, the following improvements should be made in this proposed algorithm:

- Learning parameter value should be chosen with respect to time and property of the network.
- Novel Sensing range of the sensors should be selected.

7.2 New features

This section contains about what new feature in the project can be added?. The first thing that one can add is the energy parameter to find network lifetime of the wireless sensor network. Also, there can be made further improvements in the learning algorithm with considering the appropriate rule set to obtain minimum active senors. Next one is to make more comparison between other algorithms that deal with target monitoring problem in the wireless sensor network. This comparison can be made on the basis of working time, running time, learning rate and also by varying size of terrains area.

7.3 Conclusion

This thesis focus on solving the energy issue of the wireless sensor network with solving target coverage problem. For this, there is the design of architecture and development of a machine learning algorithm. This algorithm is called adaptive learning automata algorithm. The results obtained after formulation of the experiments by using this algorithm has been analyzed to check for its validity to meet the goals of the project. Also, it has been compared with other algorithms included in related work section of the project to find efficiency of this algorithm.

This algorithm schedule sensors to save energy by providing a learning concept. Such that they can select their state to be active or sleep autonomously. And in the active state, sensors are assumed to cover the available targets using some amount of energy. Many experiments and simulation are done to evaluate the performance of the designed algorithm and architecture. The proposed algorithm provides the methodology to find the minimum active sensors that are covering all targets.

In another word it can be defined as by use of minimum active sensors, there is a use of minimum energy of the network, and thus this concept helps in conserving the energy in a wireless sensor network. So, this project addresses the concept of energy efficient target coverage network.

This algorithm applies to both small and large wireless sensor networks. But during the formulation of experiments in this type of network the learning parameter value should be considered to get precise results. Sensing range of the sensor has an equal impact on both types of networks. And, by comparing related work algorithm with concerning sensing range and number of sensors, the results show that our proposed algorithm gives efficient results than the selected algorithm for comparison.

Finally, from all the experimental results thus obtained proof that the proposed algorithm has justified the problem statement and reached the goal of the project. So, one can say that this proposed algorithm can be used as a relevant method for scheduling the sensor nodes while designing a practical sensor network.

Bibliography

[1] Ian F. Akyildiz Weilian Su Yogesh Sankarasubramaniam and Erdal Cayirci. 'A Survey On Sensor network'. In: *IEEE Communication society* 40.8 (Aug. 2002), pp. 102–114. ISSN: 0163-6804. DOI: 10.1109/MCOM.2002.1024422. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1024422.

[2] Ganesh Khadsan. *Wireless sensor network report*. URL: https://www.slideshare.net/GaneshKhadsan/wireless-sensor-network-report.

[3] Mohammad Abu Alsheikh Shaowei Lin Dusit Niyato and Hwee-Pink Tan. 'Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications'. In: *IEEE COMMUNICATION SURVEYS TUTORIALS, VOL. 16, NO. 4, FOURTH QUARTER 2014* (Apr. 2014). ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2320099. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6805162.

[4] Taiwo Oladipupo Ayodele. *Introduction to Machine Learning*. URL: http://cdn.intechweb.org/pdfs/10703.pdf.

[5] David Fumo. *Types of Machine Learning Algorithms You Should Know*. URL: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861?gi=a757b7001f4d.

[6] Mihaela Cardei My T. Thai Yingshu Li Weili Wu. 'Energy Efficient Target Coverage In Wireless Sensor Networks'. In: *IEEE Communication society* (Aug. 2005). ISSN: 0743-166X. DOI: 10.1109/INFCOM.2005.1498475. URL: https://ieeexplore.ieee.org/abstract/document/1498475.

[7] Habib Mostafaei1 Mehdi Esnaashari and Mohammad Reza Meybodi. 'A Coverage Monitoring algorithm based on Learning Automata for Wireless Sensor Networks'. In: *Cornell University Library* (Sept. 2014). URL: https://arxiv.org/ftp/arxiv/papers/1409/1409.1515.pdf.

[8] Abdul Samad Ismail Hosein Mohamadi and Shaharuddin Salleh. 'Solving Target Coverage Problem Using Cover Sets in Wireless Sensor Networks Based on Learning Automata'. In: *Wireless Personal Communications March 2014, Volume 75, Issue 1, pp 447–463* (Mar. 2014). URL: https://link.springer.com/content/pdf/10.1007%2Fs11277-013-1371-x.pdf.

[9] Habib Mostafaei and Mohammad Shojafar. 'A New Meta-heuristic Algorithm for Maximizing Lifetime of Wireless Sensor Networks'. In: *Wireless Pers Commun (2015) 82:723–742* (Jan. 2014). URL: https://link.springer.com/content/pdf/10.1007%2Fs11277-014-2249-2.pdf.

[10] Maggie X. Cheng and Xuan Gong. 'Maximum lifetime coverage preserving scheduling 105 algorithms in sensor networks'. In: Springer Science+Business Media, LLC. 2010, Sept. 2010. URL: https://link.springer.com/content/pdf/10.1007%2Fs10898-010-9636-3.pdf.

[11] Yingshu Li and Shan Gao. 'Designing k-coverage schedules in wireless sensor networks'. In: Springer Science+Business Media, LLC. 2007, Mar. 2007. URL: https://link.springer.com/content/pdf/10.1007%2Fs10878-007-9072-6.pdf.

[12] Hung-Chang Chen Kuei-Ping Shih and Bo-Jun Liu. 'Integrating Target Coverage and Connectivity for Wireless Heterogeneous Sensor Networks with Multiple Sensing Units'. In: *IEEE International Conference on networks* (2007). URL: http://tkuir.lib.tku.edu.tw/dspace/retrieve/80429/Integrating%20Target%20Coverage%20and%20Connectivity.pdf.

[13] Deepak. S. Sakkari and T. G. Basavaraju. 'Extensive Study on Coverage and Network Lifetime Issues in Wireless Sensor Networks'. In: *International Journal Of Computer Application* (Aug. 2012). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.258.5823&rep=rep1&type=pdf.

[14] M. Hefeeda and M. Bagheri. 'Randomized k-Coverage Algorithms For Dense Sensor Networks'. In: *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications* (May 2007). ISSN: 0743-166X. DOI: 10.1109/INFCOM.2007.284. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4215866.

[15] S. Mini Siba K. Udgata and Samrat L. Sabat. 'Sensor Deployment and Scheduling for Target Coverage Problem in Wireless Sensor Networks'. In: *IEEE Sensors Journal* (Mar. 2007). URL: https://ieeexplore.ieee.org/document/6637016.

[16] Chi-Fu Huang Yu-Chee Tseng and Li-Chu Lo. 'The Coverage Problem in Three-Dimensional Wireless Sensor Networks'. In: *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04* (Jan. 2005). URL: https://ieeexplore.ieee.org/abstract/document/1378938.

[17] Abdul Samad Ismail Hosein Mohamadi and Shaharuddhin Salleh. 'Utilizing distributed learning automata to solve the connected target coverage problem in directional sensor networks'. In: *IEEE Sensors Journal* (Apr. 2013). URL: https://www.sciencedirect.com/science/article/pii/S0924424713001519.

[18] Satish Chand Manju and Bijender Kumar. 'Maximising network lifetime for target coverage problem in wireless sensor networks'. In: *IET Wireless Sensor Systems* (June 2016), pp. 192–197. ISSN: 2043-6394. DOI: 10.1049/iet-wss.2015.0094. URL: https://ieeexplore.ieee.org/document/7763030.

[19] R. S. Kittur and A. N. Jadhav. 'Enhancement in Network Lifetime and Minimization of Target Coverage Problem in WSN'. In: *IEEE Sensors Journal* (Dec. 2017). DOI: 10.1109/I2CT.2017.8226308. URL: https://ieeexplore.ieee.org/document/7763030.

[20] Junkun Li Jiming Chen and Ten H. Lai. 'Trapping Mobile Targets in Wireless Sensor Networks: An Energy-Efficient Perspective'. In: *IEEE Transactions on Vehicular Technology* (Sept. 2013), pp. 3287–3300. ISSN: 1939-9359. DOI: 10.1109/TVT.2013.2254732. URL: https://ieeexplore.ieee.org/document/6488886.

[21] Satish Chand Manju and Bijender Kumar. 'Target coverage heuristic based on learning automata in wireless sensor networks'. In: *IET Wireless Sensor Systems* (Feb. 2018). ISSN: 2043-6386. DOI: 10.1049/iet-wss.2017.0090. URL: http://digital-library.theiet.org/content/journals/10.1049/iet-wss.2017.0090.

[22] Dame DIONGUE Babacar DIOP and Ousmane THIARE. 'Target Coverage Management in Wireless Sensors Networks'. In: *ICWiSe 2014, IEEE Conference on Wireless Sensors* (Nov. 2014). URL: https://www.researchgate.net/publication/266129545_Target_Coverage_Management_in_Wireless_Sensor_Networks.

[23] Mihaela Cardei and DING-ZHU DU. 'Improving Wireless Sensor Network Lifetime through Power Aware Organization'. In: *Wireless Network* (Jan. 2005). ISSN: 1022-0038. DOI: https://doi.org/10.1007/s11276-005-6615-6. URL: https://link.springer.com/content/pdf/10.1007%2Fs11276-005-6615-6.pdf.

[24] Jiang Lei Shu Guangjie Han Li Liu Jinfang and Gerhard Hancke. 'Analysis of Energy-Efficient Connected Target Coverage Algorithms for Industrial Wireless Sensor Networks'. In: *IEEE Transactions on Industrial Informatics* (Dec. 2015). ISSN: 1551-3203. DOI: 10.1109/TII.2015.2513767. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7369957.

[25] Ionut Cardei and Mihaela Cardei. 'Energy-Effcient Connected-Coverage in Wireless Sensor Networks'. In: *International Journal Of Sensor Networks* (May 2008). ISSN: 1748-1279. URL: http://www.inderscience.com/offer.php?id=18484.

[26] Panayiotis Kotzanikolaou Dimitrios Zorbas Dimitris Glynos and Christos Douligeris. 'BGOP: An Adaptive Algorithm for Coverage Problems in Wireless Sensor Networks'. In: *European Wireless* (Jan. 2007). URL: https://www.researchgate.net/publication/250849540_BGOP_An_Adaptive_Algorithm_for_Coverage_Problems_in_Wireless_Sensor_Networks.

[27] Chi-Fu Huang and Yu-Chee Tseng. 'The Coverage Problem in a Wireless Sensor Network'. In: *Mobile networks and application* (Aug. 2005). ISSN: 519–528. URL: https://link.springer.com/article/10.1007/s11036-005-1564-y.

[28] Python software Foundation. *welcome to Python.org*. 2018-9-20. Jan. 2001. URL: https://www.python.org/.

[29] Python(Programming Language). *Pyhton*. [online;accessed 11-October 2018]. URL: https://en.wikipedia.org/wiki/Python_(programming_language).

[30] Norie Fu Vorapong Suppakitpaisarn Kei Kimura and Naonori Kakimura. 'Maximum Lifetime Coverage Problems with Battery Recovery Effects'. In: *Wireless Network* (Feb. 2018). DOI: https://doi.org/10.1016/j.suscom.2018.02.007. URL: https://www.sciencedirect.com/science/article/pii/S2210537917300343.

[31] Anantha Chandrakasan Wendi Rabiner Heinzelman and Hari Balakrishnan. 'Energy-Efficient Communication Protocol for Wireless Microsensor Networks'. In: *Proceedings of the 33rd Hawaii International Conference on System Sciences - 2000* (). DOI: 10.1109/HICSS.2000.926982. URL: https://pdfs.semanticscholar.org/b383/4aed2c564d2dbeacf8c1b733dbe5c9c07fbf.pdf?_ga=2.76920341.1450741646.1540913670-1443666054.1540913670.

[32] Mohammad ali Jamali Navid Bakhshivand Mohammad Easmaeilpour and Davood Salami. 'An energy-efficient algorithm for connected target coverage problem in wireless sensor networks'. In: *2010 3rd International Conference on Computer Science and Information Technology* (Sept. 2010). DOI: 10.1109/ICCSIT.2010.5563640. URL: https://ieeexplore.ieee.org/document/5563640.

[33] D.Praveen Kumar Tarachand Amgoth & Chandra Sekhara Rao Annavarapu. 'Machine learning algorithms for wireless sensor networks: A survey'. In: *Wireless Network* (Sept. 2018). DOI: https://doi.org/10.1016/j.inffus.2018.09.013. URL: https://www.sciencedirect.com/science/article/pii/S156625351830277X?dgcid=rss_sd_all.

[34] Mohammad Reza Meybodi Habib Mostafaei and Mehdi Esnaashari. 'A Learning Automata Based Area Coverage Algorithm for Wireless Sensor Networks'. In: *Electronic Science And Technology* (Sept. 2010). URL: https://s3.amazonaws.com/academia.edu.documents/4889331/2-JEST-H067-R2ok_200-205_.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1539256731&Signature=6zpXoJD9xC48qYLcyNkRjJs%2BZTo%3D&response-content-disposition=inline%3B%20filename%3DA_Learning_Automata_Based_Area_Coverage.pdf.

[35] KUMPATI S. NARENDRA and M. A. L. THATHACHAR. 'Learning Automata A Survey'. In: (July 1974). ISSN: 0018-9472. DOI: 10.1109/TSMC.1974.5408453. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5408453.

[36] M. A. L. Thathachar and P. S. Sastry. 'Varieties of Learning Automata: An Overview'. In: *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS* (Dec. 2002). ISSN: 1941-0492. DOI: 10.1109/TSMCB.2002.1049606. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1049606.

[37] Mehdi Esnaashari and Mohammad Reza Meybodi. 'Data aggregation in sensor networks using learning automata'. In: *Wireless Networks* (Apr. 2010). ISSN: 687-699. DOI: 10.1007/s11276-009-0162-5. URL: https://www.researchgate.net/publication/226905940_Data_aggregation_in_sensor_networks_using_learning_automata.

[38] Arouna Ndam Njoya1 Christopher Thron Jordan Barry Wahabou Abdou Emmanuel Tonye Nukenine Siri Lawrencia Konje1 and Albert Dipanda. 'Efficient scalable sensor node placement algorithm for fixed target coverage applications of wireless sensor networks'. In: *IET Wireless Sensor systems* (Feb. 2017). ISSN: 2043-6386. DOI: 10.1049/iet-wss.2016.0076. URL: http://digital-library.theiet.org/content/journals/10.1049/iet-wss.2016.0076.

[39] Sonia Fahmy Yan Wu and Ness B. Shroff. 'Optimal QoS-aware Sleep/Wake Scheduling for Time-Synchronized Sensor Networks'. In: *2006 40th Annual Conference on Information Sciences and Systems* (Jan. 2007). DOI: 10.1109/CISS.2006.286599. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4067940.

[40] Yi Zou and Krishnendu Chakrabarty. 'A Distributed Coverage- and ConnectivityCentric Technique for Selecting Active Nodes in Wireless Sensor Networks'. In: *IEEE TRANSACTIONS ON COMPUTERS, VOL. 54, NO. 8* (Aug. 2005). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1453499.

[41] Ling Ding Weili WU James Willson Lidong Wu, Zaixin Lu and Wonjun Lee. 'Constant-Approximation for Target Coverage Problem in Wireless Sensor Networks'. In: *2012 Proceedings IEEE INFOCOM* (May 2012). ISSN: 0743-166X. DOI: 10.1109/INFCOM.2012.6195527. URL: https://ieeexplore.ieee.org/document/6195527.

[42] Rohit Khajuria and Sumeet Gupta. 'Energy optimization and lifetime enhancement techniques in wireless sensor networks: A Survey'. In: *International Conference on Computing, Communication Automation* (July 2015). DOI: 10.1109/CCAA.2015.7148408. URL: https://ieeexplore.ieee.org/document/7148408.

[43] Halil Yetgin Kent Tsz Kan Cheung Mohammed El-Hajjar and Lajos Hanzo. 'A Survey of Network Lifetime Maximization Techniques in Wireless Sensor Networks'. In: *IEEE Communications Surveys Tutorials* (Jan. 2017). ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2650979. URL: https://ieeexplore.ieee.org/abstract/document/7812629.

[44] Muhammad Amir Khan Halabi Hasbullah Babar Nazir and Muhammad Ali Khan. 'Lifetime and coverage maximization technique for mobile sensor networks'. In: *2014 International Conference on Computer and Information Sciences (ICCOINS)* (July 2014). DOI: 10.1109/ICCOINS.2014.6868394. URL: https://ieeexplore.ieee.org/document/6868394.

[45] Zaixin Lu Wei Wayne Li and Miao Pan. 'Maximum Lifetime Scheduling for Target Coverage and Data Collection in Wireless Sensor Networks'. In: *IEEE Transactions on Vehicular Technology* (May 2014). DOI: 10.1109/TVT.2014.2322356. URL: https://ieeexplore.ieee.org/document/6811184.

[46] Manh Thuong Quan Dao Ngoc Duy Nguyen Vyacheslav Zalyubovskiy and Hyunseung Choo1. 'An Energy-efficient Coverage Pattern of WSNs for High Rate Data Transmissions'. In: (). URL: http://worldcomp-proceedings.com/proc/p2011/ICW8239.pdf.

[47] Ines Khoufi Pascale Minet Anis Laouiti and Saoucene Mahfoudh. 'Survey of Deployment Algorithms in Wireless Sensor Networks: Coverage and Connectivity Issues and Challenges'. In: *International Journal of Autonomous and Adaptive Communications Systems (IJAACS)* (Dec. 2017). URL: https://hal.inria.fr/hal-01095749/document.

[48] D.G.Anand H.G. Chandrakanth and M.N.Giriprasad. 'An Efficient Energy, Coverage and Connectivity (Ec2) Algorithm for Wireless Sensor Networks'. In: *International Journal of Computer Applications* (May 2012). URL: https://pdfs.semanticscholar.org/0b11/a9f09b62f1f93da0702c9cd5c07856211c00.pdf.

[49] Gayarti Devi and Rajeeb Sankar Bal. 'Node Deployment Coverage in Large Wireless Sensor Networks'. In: *Journal of Network Communications and Emerging Technologies (JNCET)* (Feb. 2016). URL: https://pdfs.semanticscholar.org/e735/8f29cc3118f3a3cb909ca002b177d1dcb3c9.pdf.

[50] Shaharuddin Salleh Hosein Mohamadi and Mohd Norsyarizad Razali. 'Heuristic methods to maximize network lifetime in directional sensor networks with adjustable sensing ranges'. In: *Journal of Network and Computer Applications* (July 2014). URL: https://ac.els-cdn.com/S1084804514001866/1-s2.0-S1084804514001866-main.pdf?_tid=c03e31c7-266a-4f91-90df-7e33f4d1d521&acdnat=1541506491_a44021d55cd8f7bd5b8ef7d8f0268cdf.

[51] Valerio Persico Habib Mostafaei Antonio Montieri and Antonio Pescapé. 'A sleep scheduling approach based on learning automata for WSN partial coverage'. In: *Journal of Network and Computer Applications* (Dec. 2016). URL: https://www.sciencedirect.com/science/article/pii/S1084804516303204.

[52] Zhifei Mao Guofang Nan Guanxiong Shi and Minqiang Li. 'CDSWS: coverage-guaranteed distributed sleep/ wake scheduling for wireless sensor networks'. In: *EURASIP Journal on Wireless Communications and Networking* (Dec. 2012). URL: https://jwcn-eurasipjournals.springeropen.com/track/pdf/10.1186/1687-1499-2012-44.

[53] Dayong Ye and Minjie Zhang. ‘A Self-Adaptive Sleep/Wake-Up Scheduling Approach for Wireless Sensor Networks’. In: *IEEE TRANSACTIONS ON CYBERNETICS* (Mar. 2018). URL: https://ieeexplore-ieee-org.ezproxy.hioa.no/stamp/stamp.jsp?tp=&arnumber=7870667.

Appendices

Appendix A

Script For Experiment 1

```
import random
from math import sqrt
import matplotlib.pyplot as plt
from collections import Counter

gridsize=600
Range_sensor=150 # Take value from 50 to 600
N_sensors=20
M_Targets=10
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets

randomseed=9001


random.seed(randomseed)





if (M_Targets >N_sensors):
        print "error targets more than sensors"
        exit()


def Covering_sensors(xtarget, ytarget):
    my_list =[]
    for i in range(N_sensors):
        C= sqrt((xtarget-x_sensor[i])**2 +(ytarget-y_sensor[i])
    **2)
        if C<=Range_sensor:

            my_list.append(i)



    return my_list


```

```
def Covered_targets( xsensor,ysensor):
    my_list =[]

    for j in range(M_Targets):
        C= sqrt((x_targets[j]-xsensor)**2 +(y_targets[j]-ysensor)
    **2)
        if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by
    ", xsensor, ysensor
            my_list.append(j)

    #else:
    #   print 'no target covered'
    return my_list


def Covered_target( xsensor,ysensor,x_target, y_target):
    result=False
    #print "x tagets",x_targets
    #for j in range(M_Targets):
    C= sqrt((x_target-xsensor)**2 +(y_target-ysensor)**2)
    if C<=Range_sensor:
        #print "target", x_targets[j],y_targets[j]," coverd by",
    xsensor, ysensor
        result=True

    #else:
    #   print 'no target covered'
    return result


def Target_coverd_by_active_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets

def random_sol(LA,myrandom):
    active_sensors = []
    for i in range(N_sensors):
        r=myrandom.random()
        if r<LA[i]:
            active=1
        else:
            active=0
        #sign = randint(0,1)

        if active==1:
            active_sensors.append(1)
        else :
            active_sensors.append(0)

    return active_sensors

#Main code=
list_targets=[]
for j in range(M_Targets):

    #list_covering_sensors = []

    alreadythere=False
    while (not alreadythere):  # to avoid two targets at the same
    point
        x_targets[j] = random.randint(0, gridsize)
        y_targets[j] = random.randint(0, gridsize)
        if ([x_targets[j],y_targets[j]]) in list_targets:
            alreadythere=True
        #print x_targets[j],y_targets[j],"it is already there"
        else:
            #print "it is new"
            alreadythere=False
            list_targets.append([x_targets[j],y_targets[j]])






list_sensors=[]
#The first M sensors  should cover the first M targets
for i in range(M_Targets):

    alreadythere=False
    Covered=False
    while ((not Covered) ):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)

        Covered=Covered_target( x_sensor[i],y_sensor[i],x_targets[
    i], y_targets[i])

    #print i,  x_sensor[i],y_sensor[i],x_targets[i], y_targets[i]
    list_sensors.append([x_sensor[i],y_sensor[i]])





#The rest of sensors from M to N sesnors cover any thing

for i in range(M_Targets, N_sensors):

    list_covering_targets = []


    alreadythere=False
    while ((len(list_covering_targets)==0) or (alreadythere)):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)
        if ([x_sensor[i],y_sensor[i]]) in list_sensors:
            alreadythere=True

        else:

            alreadythere=False
```

```python
            list_covering_targets =Covered_targets(x_sensor[i],y_sensor[i])
            list_sensors.append([x_sensor[i],y_sensor[i]])





#Resetting the random number generator

all_active_sensor = []
number_iteration=[]
number_exp=1000
for t in range (number_exp):
    myrandom = random.Random()


    LA = [0.5] * N_sensors
    bestVal = 0

    minimum_sensors = N_sensors
    best_active_sensors_binary = [0] * N_sensors

    lamda = 0.01# Learning Parameter
    epsilon = 0.01

    converged = False

    iteration=0
    while (converged==False): # this loop runs till it satisfies the condition

        converged=True
        active_sensors_binary=random_sol(LA,myrandom)

        active_sensors_list=[]
        for i in range(N_sensors):
            if active_sensors_binary[i]==1:
                active_sensors_list.append(i)

        N_covered_targets=len(Target_coverd_by_active_sensors(active_sensors_list))

        number_active_sensors=len(active_sensors_list)

        if (number_active_sensors<minimum_sensors) and (N_covered_targets==M_Targets):
            minimum_sensors=number_active_sensors

            best_active_sensors_binary= active_sensors_binary[:]


        for i in  range(N_sensors):
            if LA[i]>=1-epsilon:
                LA[i]=1

            elif LA[i]<=epsilon:
                LA[i]=0

            elif (best_active_sensors_binary[i]==1):
                LA[i]=LA[i]+lamda*(1-LA[i])
            else:
                LA[i]=LA[i]+lamda*(0-LA[i])

        for i in  range(N_sensors): #This loop provide the condition to Converse the program
            if (LA[i]!=1) and (LA[i]!=0):
                converged=False

        iteration=iteration+1
    all_active_sensor.append(minimum_sensors)
    number_iteration.append(iteration)

#print "minimumsensor:",minimum_sensors

#This is for finding the sum of active sensors
"""sum = 0.0
for item in all_active_sensor:
    sum = sum + item
    Total_minimum_active_sensor = float(len(all_active_sensor))
    average= sum / n"""

print "--------------------------------------------"
print "-------------Final Result-----------------"
print"number of iterations in each experiment:", number_iteration
print"Number of Active sensors in each experiment:", all_active_sensor
#print"Numbe of occurence of element in list:",Counter(all_active_sensor) # this counts the ocurence of elements in list
print"Average number of minimum active sensor per experiment:", sum(all_active_sensor)/(number_exp*1.0)
print "sum of minimum number of sensors in all experiment:",sum(all_active_sensor)
print"Average iteration of all experiment :",sum (number_iteration)/(number_exp*1.0)
#print"Average of minimum active senors:", round(average,2) # this provide the result with two number after decimal
print "--------------------------------------------"
print "--------------------------------------------"
```

Listing A.1: Impact of sensing range of sensors

Appendix B

Script For Experiment 2

```
import random
from math import sqrt
import matplotlib.pyplot as plt
from collections import Counter

gridsize=600
Range_sensor=100  # Put this value 100,300 and 450
N_sensors=20     # Take this value from 20 to 30
M_Targets=15
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets

randomseed=9001


random.seed(randomseed)





if (M_Targets >N_sensors):
        print "error targets more than sensors"
        exit()


def Covering_sensors(xtarget, ytarget):
    my_list =[]
    for i in range(N_sensors):
        C= sqrt((xtarget-x_sensor[i])**2 +(ytarget-y_sensor[i])
    **2)
        if C<=Range_sensor:
            #print("target is covered")
            my_list.append(i)

    #else:
    #   print 'no target covered'
    return my_list


```

```
def Covered_targets( xsensor,ysensor):
    my_list =[]
    #print "x tagets",x_targets
    for j in range(M_Targets):
        C= sqrt((x_targets[j]-xsensor)**2 +(y_targets[j]-ysensor)
    **2)
        if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by
    ", xsensor, ysensor
            my_list.append(j)

    #else:
    #   print 'no target covered'
    return my_list


def Covered_target( xsensor,ysensor,x_target, y_target):
    result=False
    #print "x tagets",x_targets
    #for j in range(M_Targets):
    C= sqrt((x_target-xsensor)**2 +(y_target-ysensor)**2)
    if C<=Range_sensor:
        #print "target", x_targets[j],y_targets[j]," coverd by",
    xsensor, ysensor
        result=True

    #else:
    #   print 'no target covered'
    return result


def Target_coverd_by_active_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets

def random_sol(LA,myrandom):
    active_sensors = []
    for i in range(N_sensors):
        r=myrandom.random()
        if r<LA[i]:
            active=1
        else:
            active=0
        #sign = randint(0,1)

        if active==1:
            active_sensors.append(1)
        else :
            active_sensors.append(0)

    return active_sensors

#Main code=
list_targets=[]
for j in range(M_Targets):

    #list_covering_sensors = []

    alreadythere=False
    while (not alreadythere):  # to avoid two targets at the same
    point
        x_targets[j] = random.randint(0, gridsize)
        y_targets[j] = random.randint(0, gridsize)
        if ([x_targets[j],y_targets[j]]) in list_targets:
            alreadythere=True
        #print x_targets[j],y_targets[j],"it is already there"
        else:
            #print "it is new"
            alreadythere=False
            list_targets.append([x_targets[j],y_targets[j]])






list_sensors=[]
#The first M sensors  should cover the first M targets
for i in range(M_Targets):

    alreadythere=False
    Covered=False
    while ((not Covered) ):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)

        Covered=Covered_target( x_sensor[i],y_sensor[i],x_targets[
    i], y_targets[i])

    #print i,  x_sensor[i],y_sensor[i],x_targets[i], y_targets[i]
    list_sensors.append([x_sensor[i],y_sensor[i]])






#The rest of sensors from M to N sesnors cover any thing

for i in range(M_Targets, N_sensors):

    list_covering_targets = []


    alreadythere=False
    while ((len(list_covering_targets)==0) or (alreadythere)):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)
        if ([x_sensor[i],y_sensor[i]]) in list_sensors:
            alreadythere=True

        else:

            alreadythere=False
            list_covering_targets =Covered_targets(x_sensor[i],
    y_sensor[i])
            list_sensors.append([x_sensor[i],y_sensor[i]])





#Resetting the random number generator

all_active_sensor = []
number_iteration=[]
number_exp=1000
for t in range (number_exp):
    myrandom = random.Random()


    LA = [0.5] * N_sensors
    bestVal = 0

    minimum_sensors = N_sensors
    best_active_sensors_binary = [0] * N_sensors

    lamda = 0.01# Learning Parameter
    epsilon = 0.01

    converged = False

    iteration=0
    while (converged==False): # this loop runs till it satisfies
    the condition

        converged=True
        active_sensors_binary=random_sol(LA,myrandom)

        active_sensors_list=[]
        for i in range(N_sensors):
            if active_sensors_binary[i]==1:
                active_sensors_list.append(i)

        N_covered_targets=len(Target_coverd_by_active_sensors(
    active_sensors_list))

        number_active_sensors=len(active_sensors_list)

        if (number_active_sensors<minimum_sensors) and (
    N_covered_targets==M_Targets):
            minimum_sensors=number_active_sensors

            best_active_sensors_binary= active_sensors_binary[:]


        for i in  range(N_sensors):
            if LA[i]>=1-epsilon:
                LA[i]=1

            elif LA[i]<=epsilon:
                LA[i]=0

            elif (best_active_sensors_binary[i]==1):
                LA[i]=LA[i]+lamda*(1-LA[i])
            else:
                LA[i]=LA[i]+lamda*(0-LA[i])

        for i in  range(N_sensors): #This loop provide the condition to Converse the program
            if (LA[i]!=1) and (LA[i]!=0):
                converged=False

        iteration=iteration+1
    all_active_sensor.append(minimum_sensors)
    number_iteration.append(iteration)

#print "minimumsensor:",minimum_sensors

#This is for finding the sum of active sensors
"""sum = 0.0
for item in all_active_sensor:
    sum = sum + item
    Total_minimum_active_sensor = float(len(all_active_sensor))
    average= sum / n"""

print "----------------------------------------"
print "------------Final Result----------------"
print"number of iterations in each experiment:", number_iteration
print"Number of Active sensors in each experiment:", all_active_sensor
#print"Numbe of occurence of element in list:",Counter(all_active_sensor) # this counts the ocurence of elements in list
print"Average number of minimum active sensor per experiment:", sum(all_active_sensor)/(number_exp*1.0)
print "sum of minimum number of sensors in all experiment:",sum(all_active_sensor)
print"Average iteration of all experiment :",sum (number_iteration)/(number_exp*1.0)
#print"Average of minimum active senors:", round(average,2) # this provide the result with two number after decimal
print "----------------------------------------"
print "----------------------------------------"
```

Listing B.1: Impact of density of sensor by taking sensors between 19 to 31

Appendix C

Script Of Experiment 3

```python
import random
from math import sqrt
import matplotlib.pyplot as plt
from collections import Counter

gridsize=600
Range_sensor=100          # Take this value 100 and 300
N_sensors=30
M_Targets=15
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets

randomseed=9001


random.seed(randomseed)





if (M_Targets >N_sensors):
        print "error targets more than sensors"
        exit()


def Covering_sensors(xtarget, ytarget):
    my_list =[]
    for i in range(N_sensors):
        C= sqrt((xtarget-x_sensor[i])**2 +(ytarget-y_sensor[i])**2)
        if C<=Range_sensor:
            #print("target is covered")
            my_list.append(i)

    #else:
    #   print 'no target covered'
    return my_list


```

```
def Covered_targets( xsensor,ysensor):
    my_list =[]
    #print "x tagets",x_targets
    for j in range(M_Targets):
        C= sqrt((x_targets[j]-xsensor)**2 +(y_targets[j]-ysensor)
    **2)
        if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by
    ", xsensor, ysensor
            my_list.append(j)

    #else:
    #   print 'no target covered'
    return my_list


def Covered_target( xsensor,ysensor,x_target, y_target):
    result=False
    #print "x tagets",x_targets
    #for j in range(M_Targets):
    C= sqrt((x_target-xsensor)**2 +(y_target-ysensor)**2)
    if C<=Range_sensor:
        #print "target", x_targets[j],y_targets[j]," coverd by",
    xsensor, ysensor
        result=True

    #else:
    #   print 'no target covered'
    return result


def Target_coverd_by_active_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets

def random_sol(LA,myrandom):
    active_sensors = []
    for i in range(N_sensors):
        r=myrandom.random()
        if r<LA[i]:
            active=1
        else:
            active=0
        #sign = randint(0,1)

        if active==1:
            active_sensors.append(1)
        else :
            active_sensors.append(0)

    return active_sensors

#Main code=
list_targets=[]
for j in range(M_Targets):

    #list_covering_sensors = []

    alreadythere=False
    while (not alreadythere):  # to avoid two targets at the same
    point
        x_targets[j] = random.randint(0, gridsize)
        y_targets[j] = random.randint(0, gridsize)
        if ([x_targets[j],y_targets[j]]) in list_targets:
            alreadythere=True
        #print x_targets[j],y_targets[j],"it is already there"
        else:
            #print "it is new"
            alreadythere=False
            list_targets.append([x_targets[j],y_targets[j]])






list_sensors=[]
#The first M sensors  should cover the first M targets
for i in range(M_Targets):

    alreadythere=False
    Covered=False
    while ((not Covered) ):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)

        Covered=Covered_target( x_sensor[i],y_sensor[i],x_targets[
    i], y_targets[i])

    #print i,  x_sensor[i],y_sensor[i],x_targets[i], y_targets[i]
    list_sensors.append([x_sensor[i],y_sensor[i]])





#The rest of sensors from M to N sesnors cover any thing

for i in range(M_Targets, N_sensors):

    list_covering_targets = []


    alreadythere=False
    while ((len(list_covering_targets)==0) or (alreadythere)):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)
        if ([x_sensor[i],y_sensor[i]]) in list_sensors:
            alreadythere=True

        else:

            alreadythere=False
```

```python
            list_covering_targets =Covered_targets(x_sensor[i],y_sensor[i])
            list_sensors.append([x_sensor[i],y_sensor[i]])






#Resetting the random number generator

all_active_sensor = []
number_iteration=[]
number_exp=1000
for t in range (number_exp):
    myrandom = random.Random()


    LA = [0.5] * N_sensors
    bestVal = 0

    minimum_sensors = N_sensors
    best_active_sensors_binary = [0] * N_sensors

    lamda = 0.01# Learning Parameter
    epsilon = 0.01

    converged = False

    iteration=0
    while (converged==False): # this loop runs till it satisfies the condition

        converged=True
        active_sensors_binary=random_sol(LA,myrandom)

        active_sensors_list=[]
        for i in range(N_sensors):
            if active_sensors_binary[i]==1:
                active_sensors_list.append(i)

        N_covered_targets=len(Target_coverd_by_active_sensors(active_sensors_list))

        number_active_sensors=len(active_sensors_list)

        if (number_active_sensors<minimum_sensors) and (N_covered_targets==M_Targets):
            minimum_sensors=number_active_sensors

            best_active_sensors_binary= active_sensors_binary[:]


        for i in  range(N_sensors):
            if LA[i]>=1-epsilon:
                LA[i]=1

            elif LA[i]<=epsilon:
                LA[i]=0

            elif (best_active_sensors_binary[i]==1):
                LA[i]=LA[i]+lamda*(1-LA[i])
            else:
                LA[i]=LA[i]+lamda*(0-LA[i])

        for i in  range(N_sensors): #This loop provide the condition to Converse the program
            if (LA[i]!=1) and (LA[i]!=0):
                converged=False

        iteration=iteration+1
    all_active_sensor.append(minimum_sensors)
    number_iteration.append(iteration)

#print "minimumsensor:",minimum_sensors

#This is for finding the sum of active sensors
"""sum = 0.0
for item in all_active_sensor:
    sum = sum + item
    Total_minimum_active_sensor = float(len(all_active_sensor))
    average= sum / n"""

print "----------------------------------------"
print "------------Final Result----------------"
print"number of iterations in each experiment:", number_iteration
print"Number of Active sensors in each experiment:", all_active_sensor
#print"Number of occurrence of element in list:",Counter(all_active_sensor) # this counts the occurrence of elements in list
print"Average number of minimum active sensor per experiment:", sum(all_active_sensor)/(number_exp*1.0)
print "sum of minimum number of sensors in all experiment:",sum(all_active_sensor)
print"Average iteration of all experiment :",sum (number_iteration)/(number_exp*1.0)
#print"Average of minimum active senors:", round(average,2) # this provide the result with two number after decimal
print "----------------------------------------"
print "----------------------------------------"
```

Listing C.1: Code for impact of density of targets

Appendix D

Script Of Experiment 4

```
 random
from math import sqrt
import matplotlib.pyplot as plt
from collections import Counter

gridsize=600
Range_sensor=300          # Take this value to 300 and 450
N_sensors=70              # Take the values 70 to 80
M_Targets=50
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets

randomseed=9001


random.seed(randomseed)





if (M_Targets >N_sensors):
        print "error targets more than sensors"
        exit()


def Covering_sensors(xtarget, ytarget):
    my_list =[]
    for i in range(N_sensors):
        C= sqrt((xtarget-x_sensor[i])**2 +(ytarget-y_sensor[i])
    **2)
        if C<=Range_sensor:
            #print("target is covered")
            my_list.append(i)

    #else:
    #   print 'no target covered'
    return my_list


```

```
def Covered_targets( xsensor,ysensor):
    my_list =[]
    #print "x tagets",x_targets
    for j in range(M_Targets):
        C= sqrt((x_targets[j]-xsensor)**2 +(y_targets[j]-ysensor)
    **2)
        if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by
    ", xsensor, ysensor
            my_list.append(j)

    #else:
    #   print 'no target covered'
    return my_list


def Covered_target( xsensor,ysensor,x_target, y_target):
    result=False
    #print "x tagets",x_targets
    #for j in range(M_Targets):
    C= sqrt((x_target-xsensor)**2 +(y_target-ysensor)**2)
    if C<=Range_sensor:
        #print "target", x_targets[j],y_targets[j]," coverd by",
    xsensor, ysensor
        result=True

    #else:
    #   print 'no target covered'
    return result


def Target_coverd_by_active_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets

def random_sol(LA,myrandom):
    active_sensors = []
    for i in range(N_sensors):
        r=myrandom.random()
        if r<LA[i]:
            active=1
        else:
            active=0
        #sign = randint(0,1)

        if active==1:
            active_sensors.append(1)
        else :
            active_sensors.append(0)

    return active_sensors

#Main code=
list_targets=[]
for j in range(M_Targets):

    #list_covering_sensors = []

    alreadythere=False
    while (not alreadythere):  # to avoid two targets at the same
    point
        x_targets[j] = random.randint(0, gridsize)
        y_targets[j] = random.randint(0, gridsize)
        if ([x_targets[j],y_targets[j]]) in list_targets:
            alreadythere=True
        #print x_targets[j],y_targets[j],"it is already there"
        else:
            #print "it is new"
            alreadythere=False
            list_targets.append([x_targets[j],y_targets[j]])






list_sensors=[]
#The first M sensors  should cover the first M targets
for i in range(M_Targets):

    alreadythere=False
    Covered=False
    while ((not Covered) ):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)

        Covered=Covered_target( x_sensor[i],y_sensor[i],x_targets[
    i], y_targets[i])

    #print i,  x_sensor[i],y_sensor[i],x_targets[i], y_targets[i]
    list_sensors.append([x_sensor[i],y_sensor[i]])





#The rest of sensors from M to N sesnors cover any thing

for i in range(M_Targets, N_sensors):

    list_covering_targets = []


    alreadythere=False
    while ((len(list_covering_targets)==0) or (alreadythere)):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)
        if ([x_sensor[i],y_sensor[i]]) in list_sensors:
            alreadythere=True

        else:

            alreadythere=False
```

```python
            list_covering_targets =Covered_targets(x_sensor[i],y_sensor[i])
            list_sensors.append([x_sensor[i],y_sensor[i]])





#Resetting the random number generator

all_active_sensor = []
number_iteration=[]
number_exp=1000
for t in range (number_exp):
    myrandom = random.Random()


    LA = [0.5] * N_sensors
    bestVal = 0

    minimum_sensors = N_sensors
    best_active_sensors_binary = [0] * N_sensors

    lamda = 0.004# Learning Parameter
    epsilon = 0.01

    converged = False

    iteration=0
    while (converged==False): # this loop runs till it satisfies the condition

        converged=True
        active_sensors_binary=random_sol(LA,myrandom)

        active_sensors_list=[]
        for i in range(N_sensors):
            if active_sensors_binary[i]==1:
                active_sensors_list.append(i)

        N_covered_targets=len(Target_coverd_by_active_sensors(active_sensors_list))

        number_active_sensors=len(active_sensors_list)

        if (number_active_sensors<minimum_sensors) and (N_covered_targets==M_Targets):
            minimum_sensors=number_active_sensors

            best_active_sensors_binary= active_sensors_binary[:]


        for i in  range(N_sensors):
            if LA[i]>=1-epsilon:
                LA[i]=1

            elif LA[i]<=epsilon:
                LA[i]=0

            elif (best_active_sensors_binary[i]==1):
                LA[i]=LA[i]+lamda*(1-LA[i])
            else:
                LA[i]=LA[i]+lamda*(0-LA[i])

        for i in  range(N_sensors): #This loop provide the condition to Converse the program
            if (LA[i]!=1) and (LA[i]!=0):
                converged=False

        iteration=iteration+1
    all_active_sensor.append(minimum_sensors)
    number_iteration.append(iteration)

#print "minimumsensor:",minimum_sensors

#This is for finding the sum of active sensors
"""sum = 0.0
for item in all_active_sensor:
    sum = sum + item
    Total_minimum_active_sensor = float(len(all_active_sensor))
    average= sum / n"""

print "----------------------------------------------"
print "-------------Final Result-----------------"
print"number of iterations in each experiment:", number_iteration
print"Number of Active sensors in each experiment:", all_active_sensor
#print"Numbe of occurence of element in list:",Counter(all_active_sensor) # this counts the ocurence of elements in list
print"Average number of minimum active sensor per experiment:", sum(all_active_sensor)/(number_exp*1.0)
print "sum of minimum number of sensors in all experiment:",sum(all_active_sensor)
print"Average iteration of all experiment :",sum (number_iteration)/(number_exp*1.0)
#print"Average of minimum active senors:", round(average,2) # this provide the result with two number after decimal
print "----------------------------------------------"
print "----------------------------------------------"
```

Listing D.1: Impact of sensor sensing range in large network

Appendix E

Script Of Experiment 5

```
import random
from math import sqrt
import matplotlib.pyplot as plt
from collections import Counter

gridsize=600
Range_sensor=200          # Take this value from 50 to 400
N_sensors=  20         # Take the values 10 to 25
M_Targets=9
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets

randomseed=9001


random.seed(randomseed)





if (M_Targets >N_sensors):
        print "error targets more than sensors"
        exit()


def Covering_sensors(xtarget, ytarget):
    my_list =[]
    for i in range(N_sensors):
        C= sqrt((xtarget-x_sensor[i])**2 +(ytarget-y_sensor[i])
    **2)
        if C<=Range_sensor:
            #print("target is covered")
            my_list.append(i)

    #else:
    #    print 'no target covered'
    return my_list

```

```
def Covered_targets( xsensor,ysensor):
    my_list =[]
    #print "x tagets",x_targets
    for j in range(M_Targets):
        C= sqrt((x_targets[j]-xsensor)**2 +(y_targets[j]-ysensor)
    **2)
        if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by
    ", xsensor, ysensor
            my_list.append(j)

    #else:
    #   print 'no target covered'
    return my_list


def Covered_target( xsensor,ysensor,x_target, y_target):
    result=False
    #print "x tagets",x_targets
    #for j in range(M_Targets):
    C= sqrt((x_target-xsensor)**2 +(y_target-ysensor)**2)
    if C<=Range_sensor:
        #print "target", x_targets[j],y_targets[j]," coverd by",
    xsensor, ysensor
        result=True

    #else:
    #   print 'no target covered'
    return result


def Target_coverd_by_active_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets

def random_sol(LA,myrandom):
    active_sensors = []
    for i in range(N_sensors):
        r=myrandom.random()
        if r<LA[i]:
            active=1
        else:
            active=0
        #sign = randint(0,1)

        if active==1:
            active_sensors.append(1)
        else :
            active_sensors.append(0)

    return active_sensors

#Main code=
list_targets=[]
for j in range(M_Targets):

    #list_covering_sensors = []

    alreadythere=False
    while (not alreadythere):  # to avoid two targets at the same
    point
        x_targets[j] = random.randint(0, gridsize)
        y_targets[j] = random.randint(0, gridsize)
        if ([x_targets[j],y_targets[j]]) in list_targets:
            alreadythere=True
        #print x_targets[j],y_targets[j],"it is already there"
        else:
            #print "it is new"
            alreadythere=False
            list_targets.append([x_targets[j],y_targets[j]])






list_sensors=[]
#The first M sensors  should cover the first M targets
for i in range(M_Targets):

    alreadythere=False
    Covered=False
    while ((not Covered) ):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)

        Covered=Covered_target( x_sensor[i],y_sensor[i],x_targets[
    i], y_targets[i])

    #print i,  x_sensor[i],y_sensor[i],x_targets[i], y_targets[i]
    list_sensors.append([x_sensor[i],y_sensor[i]])





#The rest of sensors from M to N sesnors cover any thing

for i in range(M_Targets, N_sensors):

    list_covering_targets = []


    alreadythere=False
    while ((len(list_covering_targets)==0) or (alreadythere)):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)
        if ([x_sensor[i],y_sensor[i]]) in list_sensors:
            alreadythere=True

        else:

            alreadythere=False
            list_covering_targets =Covered_targets(x_sensor[i],
    y_sensor[i])
            list_sensors.append([x_sensor[i],y_sensor[i]])





#Resetting the random number generator

all_active_sensor = []
number_iteration=[]
number_exp=1000
for t in range (number_exp):
    myrandom = random.Random()


    LA = [0.5] * N_sensors
    bestVal = 0

    minimum_sensors = N_sensors
    best_active_sensors_binary = [0] * N_sensors

    lamda = 0.01 # Learning Parameter
    epsilon = 0.01

    converged = False

    iteration=0
    while (converged==False): # this loop runs till it satisfies
    the condition

        converged=True
        active_sensors_binary=random_sol(LA,myrandom)

        active_sensors_list=[]
        for i in range(N_sensors):
            if active_sensors_binary[i]==1:
                active_sensors_list.append(i)

        N_covered_targets=len(Target_coverd_by_active_sensors(
    active_sensors_list))

        number_active_sensors=len(active_sensors_list)

        if (number_active_sensors<minimum_sensors) and (
    N_covered_targets==M_Targets):
            minimum_sensors=number_active_sensors

            best_active_sensors_binary= active_sensors_binary[:]


        for i in  range(N_sensors):
            if LA[i]>=1-epsilon:
                LA[i]=1

            elif LA[i]<=epsilon:
                LA[i]=0

            elif (best_active_sensors_binary[i]==1):
                LA[i]=LA[i]+lamda*(1-LA[i])
            else:
                LA[i]=LA[i]+lamda*(0-LA[i])

        for i in  range(N_sensors): #This loop provide the condition to Converse the program
            if (LA[i]!=1) and (LA[i]!=0):
                converged=False

        iteration=iteration+1
    all_active_sensor.append(minimum_sensors)
    number_iteration.append(iteration)

#print "minimumsensor:",minimum_sensors

#This is for finding the sum of active sensors
"""sum = 0.0
for item in all_active_sensor:
    sum = sum + item
    Total_minimum_active_sensor = float(len(all_active_sensor))
    average= sum / n"""

print "----------------------------------------------------------------"
print "--------------------Final Result--------------------------"
print"number of iterations in each experiment:", number_iteration
print"Number of Active sensors in each experiment:", all_active_sensor
#print"Numbe of occurence of element in list:",Counter(all_active_sensor) # this counts the ocurence of elements in list
print"Average number of minimum active sensor per experiment:", sum(all_active_sensor)/(number_exp*1.0)
print "sum of minimum number of sensors in all experiment:",sum(all_active_sensor)
print"Average iteration of all experiment :",sum (number_iteration)/(number_exp*1.0)
#print"Average of minimum active senors:", round(average,2) # this provide the result with two number after decimal
print "----------------------------------------------------------------"
print "----------------------------------------------------------------"
```

Listing E.1: Impact of learning parameter

Appendix F

Script Of Experiment 6

```python
import random
from math import sqrt
import matplotlib.pyplot as plt
from collections import Counter

gridsize=600
Range_sensor=200 # This value is constant
N_sensors=10     # This value ranges from 10 to 25
M_Targets=9
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets

randomseed=9001


random.seed(randomseed)





if (M_Targets >N_sensors):
        print "error targets more than sensors"
        exit()


def Covering_sensors(xtarget, ytarget):
    my_list =[]
    for i in range(N_sensors):
        C= sqrt((xtarget-x_sensor[i])**2 +(ytarget-y_sensor[i])
    **2)
        if C<=Range_sensor:
            #print("target is covered")
            my_list.append(i)

    #else:
    #   print 'no target covered'
    return my_list


```

```
def Covered_targets( xsensor,ysensor):
    my_list =[]
    #print "x tagets",x_targets
    for j in range(M_Targets):
        C= sqrt((x_targets[j]-xsensor)**2 +(y_targets[j]-ysensor)
    **2)
        if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by
    ", xsensor, ysensor
            my_list.append(j)

    #else:
    #   print 'no target covered'
    return my_list


def Covered_target( xsensor,ysensor,x_target, y_target):
    result=False
    #print "x tagets",x_targets
    #for j in range(M_Targets):
    C= sqrt((x_target-xsensor)**2 +(y_target-ysensor)**2)
    if C<=Range_sensor:
        #print "target", x_targets[j],y_targets[j]," coverd by",
    xsensor, ysensor
        result=True

    #else:
    #   print 'no target covered'
    return result


def Target_coverd_by_active_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets

def random_sol(LA,myrandom):
    active_sensors = []
    for i in range(N_sensors):
        r=myrandom.random()
        if r<LA[i]:
            active=1
        else:
            active=0
        #sign = randint(0,1)

        if active==1:
            active_sensors.append(1)
        else :
            active_sensors.append(0)

    return active_sensors

#Main code=
list_targets=[]
for j in range(M_Targets):

    #list_covering_sensors = []

    alreadythere=False
    while (not alreadythere):  # to avoid two targets at the same
    point
        x_targets[j] = random.randint(0, gridsize)
        y_targets[j] = random.randint(0, gridsize)
        if ([x_targets[j],y_targets[j]]) in list_targets:
            alreadythere=True
        #print x_targets[j],y_targets[j],"it is already there"
        else:
            #print "it is new"
            alreadythere=False
            list_targets.append([x_targets[j],y_targets[j]])






list_sensors=[]
#The first M sensors  should cover the first M targets
for i in range(M_Targets):

    alreadythere=False
    Covered=False
    while ((not Covered) ):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)

        Covered=Covered_target( x_sensor[i],y_sensor[i],x_targets[
    i], y_targets[i])

    #print i,  x_sensor[i],y_sensor[i],x_targets[i], y_targets[i]
    list_sensors.append([x_sensor[i],y_sensor[i]])






#The rest of sensors from M to N sesnors cover any thing

for i in range(M_Targets, N_sensors):

    list_covering_targets = []


    alreadythere=False
    while ((len(list_covering_targets)==0) or (alreadythere)):
        x_sensor[i] = random.randint(0, gridsize)
        y_sensor[i] = random.randint(0, gridsize)
        if ([x_sensor[i],y_sensor[i]]) in list_sensors:
            alreadythere=True

        else:

            alreadythere=False
            list_covering_targets =Covered_targets(x_sensor[i],
    y_sensor[i])
            list_sensors.append([x_sensor[i],y_sensor[i]])





#Resetting the random number generator

all_active_sensor = []
number_iteration=[]
number_exp=1000
for t in range (number_exp):
    myrandom = random.Random()


    LA = [0.5] * N_sensors
    bestVal = 0

    minimum_sensors = N_sensors
    best_active_sensors_binary = [0] * N_sensors

    lamda = 0.01# Learning Parameter (change this value)
    epsilon = 0.01

    converged = False

    iteration=0
    while (converged==False): # this loop runs till it satisfies
    the condition

        converged=True
        active_sensors_binary=random_sol(LA,myrandom)

        active_sensors_list=[]
        for i in range(N_sensors):
            if active_sensors_binary[i]==1:
                active_sensors_list.append(i)

        N_covered_targets=len(Target_coverd_by_active_sensors(
    active_sensors_list))

        number_active_sensors=len(active_sensors_list)

        if (number_active_sensors<minimum_sensors) and (
    N_covered_targets==M_Targets):
            minimum_sensors=number_active_sensors

            best_active_sensors_binary= active_sensors_binary[:]


        for i in  range(N_sensors):
            if LA[i]>=1-epsilon:
                LA[i]=1

            elif LA[i]<=epsilon:
                LA[i]=0

            elif (best_active_sensors_binary[i]==1):
                LA[i]=LA[i]+lamda*(1-LA[i])
            else:
                LA[i]=LA[i]+lamda*(0-LA[i])

        for i in  range(N_sensors): #This loop provide the condition to Converse the program
            if (LA[i]!=1) and (LA[i]!=0):
                converged=False

        iteration=iteration+1
    all_active_sensor.append(minimum_sensors)
    number_iteration.append(iteration)

#print "minimumsensor:",minimum_sensors

#This is for finding the sum of active sensors
"""sum = 0.0
for item in all_active_sensor:
    sum = sum + item
    Total_minimum_active_sensor = float(len(all_active_sensor))
    average= sum / n"""

print "--------------------------------------------------"
print "--------------Final Result--------------------"
print"number of iterations in each experiment:", number_iteration
print"Number of Active sensors in each experiment:", all_active_sensor
#print"Numbe of occurence of element in list:",Counter(all_active_sensor) # this counts the ocurence of elements in list
print"Average number of minimum active sensor per experiment:", sum(all_active_sensor)/(number_exp*1.0)
print "sum of minimum number of sensors in all experiment:",sum(all_active_sensor)
print"Average iteration of all experiment :",sum (number_iteration)/(number_exp*1.0)
print"minimum value:",min(all_active_sensor)
print"max:",max(all_active_sensor)
#print"Average of minimum active senors:", round(average,2) # this provide the result with two number after decimal
print "--------------------------------------------------"
print "--------------------------------------------------"
```

Listing F.1: Comparison of result obtained from algorithm with brute force method

Appendix G

Script For Experiment 7

```
import random
from math import sqrt
import matplotlib.pyplot as plt


gridsize=600
Range_sensor= 300
N_sensors=5
M_Targets=2
x_sensor=[0]*N_sensors
y_sensor=[0]*N_sensors
x_targets=[0]*M_Targets
y_targets=[0]*M_Targets
randomseed=9001


random.seed(randomseed)



LA=[0.5]*N_sensors
bestVal=0


mimimum_sensors=N_sensors
best_active_sensors_binary=[0]*N_sensors


lamda=0.01
epsilon=0.01
beta=0

converged=False



if (M_Targets>N_sensors):
    print "error targets more than sensors"
    exit()


def Covering_sensors(xtarget, ytarget):
    my_list =[]
    for i in range(N_sensors):
        C= sqrt((xtarget-x_sensor[i])**2 +(ytarget-y_sensor[i])**2)
        if C<=Range_sensor:

            my_list.append(i)


    return my_list


def Covered_targets( xsensor,ysensor):
    my_list =[]
    #print "x tagets",x_targets
    for j in range(M_Targets):
        C= sqrt((x_targets[j]-xsensor)**2 +(y_targets[j]-ysensor)**2)
        if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by ", xsensor, ysensor
            my_list.append(j)

        #else:
         #   print 'no target covered'
    return my_list


def Covered_target( xsensor,ysensor,x_target, y_target):
    result=False
    #print "x tagets",x_targets
    #for j in range(M_Targets):
    C= sqrt((x_target-xsensor)**2 +(y_target-ysensor)**2)
    if C<=Range_sensor:
            #print "target", x_targets[j],y_targets[j]," coverd by ", xsensor, ysensor
            result=True

        #else:
         #   print 'no target covered'
    return result


def Targest_coverd_by_active_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets


def Targets_coverd_by_list_sensors(Active_Sensors):
    Set_Covered_Targets=set()
    for i in Active_Sensors:
        target_covered= Covered_targets(x_sensor[i],y_sensor[i])
        Set_Covered_Targets.update(set(target_covered))

    return Set_Covered_Targets
```

```python



def random_sol(LA,myrandom):
  active_sensors = []
  for i in range(N_sensors):
    r=myrandom.random()
    if r<LA[i]:
      active=1
    else:
      active=0
    #sign = randint(0,1)

    if active==1:
      active_sensors.append(1)
    else :
      active_sensors.append(0)

  return active_sensors

#Main code
list_targets=[]
for j in range(M_Targets):

        #list_covering_sensors = []

        alreadythere=False
        while (not alreadythere):  # to make sure each target is
    coverged by at least one sensor
            x_targets[j] = random.randint(0, gridsize)
            y_targets[j] = random.randint(0, gridsize)
            if ([x_targets[j],y_targets[j]]) in list_targets:
                alreadythere=True
                #print x_targets[j],y_targets[j],"it is already
    there"
            else:
                #print "it is new"
                alreadythere=False
                list_targets.append([x_targets[j],y_targets[j]])

            x2 = x_targets[j]
            y2 = y_targets[j]

        # plt.scatter(x2, y2, color='red')
 #          plt.scatter(x2, y2, 100, 'g', '^')

#            plt.text(x2 * (1 + 0.01), y2 * (1 + 0.01), j,
    fontsize=12)




list_sensors=[]
#The first M sensors  should cover the first M targets
for i in range(M_Targets):


    alreadythere=False
    Covered=False
    while ((not Covered) ):
            x_sensor[i] = random.randint(0, gridsize)
            y_sensor[i] = random.randint(0, gridsize)
            #print "here", x_sensor[i],y_sensor[i]
            #if ([x_sensor[i],y_sensor[i]]) in list_sensors:
                #alreadythere=True
            Covered=Covered_target( x_sensor[i],y_sensor[i], x_targets[i], y_targets[i])
                #print x_targets[j],y_targets[j],"it is already there"
            #else:
                #print "it is new"
                #alreadythere=False
    print i,  x_sensor[i],y_sensor[i],x_targets[i], y_targets[i]
    list_sensors.append([x_sensor[i],y_sensor[i]])

                #Covering_sensors(x_targets[i], x_targets[i])



#The rest of sensors from M to N sesnors cover any thing

for i in range(M_Targets, N_sensors):

        list_covering_targets = []

        alreadythere=False
        while ((len(list_covering_targets)==0) or (alreadythere)):
            x_sensor[i] = random.randint(0, gridsize)
            y_sensor[i] = random.randint(0, gridsize)
            if ([x_sensor[i],y_sensor[i]]) in list_sensors:
                alreadythere=True
                #print x_targets[j],y_targets[j],"it is already there"
            else:
                #print "it is new"
                alreadythere=False
                list_covering_targets =Covered_targets(x_sensor[i], y_sensor[i])
            #print "covered and not there ?", (list_covering_targets==[]) and (not alreadythere)
            #print "already there ?", not alreadythere
        print "exited, sensor covering, ",list_covering_targets

            #print("sensor", i, "is at position", x_sensor[i], y_sensor[i])
        list_sensors.append([x_sensor[i],y_sensor[i]])

        #print "sensor", i, "is covering those targets", list_covering_targets
        x1 = x_sensor[i]
        y1 = y_sensor[i]
 #       plt.text(x1 * (1 + 0.01), y1 * (1 + 0.01), i, fontsize=12)  # code for ploting the points with text

        plt.scatter(x1, y1, 100, 'r', '1')  # for scatter plot of sensors
        circle = plt.Circle((x1, y1), Range_sensor, color='b',fill=False)  # this code is for drawing circle coverage of sensor
        fig = plt.gcf()
        fig.gca().add_artist(circle)


 #    plt.ylim(0, 100)
 #    plt.xlim(0, 100)

#Active_Sensors=[1]



#print "COVERGED TARGETS BY ACTIVE SET SENSORS",
    Targest_coverd_by_active_sensors(Active_Sensors)




#Resetting the random number generator
myrandom = random.Random()

iteration=0
while (converged==False):

    converged=True
    active_sensors_binary=random_sol(LA,myrandom)

    active_sensors_list=[]


    for i in range(N_sensors):
        Only_me_covering=False
        sensor=[i]
        other_active_sensors=[k for k in active_sensors_binary
    if k  != i]
        mytargets=Targets_coverd_by_list_sensors(sensor)
        targets_of_others=Targets_coverd_by_list_sensors(
    other_active_sensors)
        if mytargets.intersection(targets_of_others):
            Only_me_covering=False
        else:
            Only_me_covering=True

        if active_sensors_binary[i]==1:
                #print "Anis"
                #print Only_me_covering
                active_sensors_list.append(i)
                if Only_me_covering==True:
                     LA[i]=LA[i]+beta*(1-LA[i])
                #else:
                    # LA[i]=LA[i]+lamda*(0-LA[i])
        else:#Sensor inactive
               if Only_me_covering==False:#Means other cover when
    sleep
                     LA[i]=LA[i]+beta*(0-LA[i])
               #else:
#                    LA[i]=LA[i]+lamda*(1-LA[i])



```

```python

    N_covered_targets=len(Targest_coverd_by_active_sensors(active_sensors_list))

    #print "random active", active_sensors_list, "covers a number of targest =",   N_covered_targets

    number_active_sensors=len(active_sensors_list)
    #for i in active_sensors_binary:
#        if i==1:
#            number_active_sensors=number_active_sensors+1






    #if (number_active_sensors<mimimum_sensors) and (N_covered_targets==M_Targets):
#        mimimum_sensors=number_active_sensors
#        best_active_sensors_binary=   active_sensors_binary[:]


    for i in  range(N_sensors):
        if LA[i]>=1-epsilon:
            LA[i]=1

        elif LA[i]<=epsilon:
            LA[i]=0



    for i in  range(N_sensors):
        if (LA[i]!=1) and (LA[i]!=0):
            converged=False

    iteration=iteration+1

    print "best active", best_active_sensors_binary
    print "mimimum_sensors", mimimum_sensors
    print "LA vector", LA
    print "iterations", iteration

plt.scatter(x_sensor, y_sensor, 100, "g", "1")

for i in range(N_sensors):
    if best_active_sensors_binary[i]==1:
        circle = plt.Circle((x_sensor[i], y_sensor[i]), Range_sensor, color='b',fill=False)  # this code is for drawing circle coverage of sensor
        fig = plt.gcf()
        fig.gca().add_artist(circle)


plt.scatter(x_targets, y_targets, 100, "r", "1")
plt.ylim(0,gridsize)
plt.xlim(0,gridsize)
plt.show()
```

Listing G.1: Code of algorithm that has been compared with proposed algorithm