ECE 659 Assignment 2

Group 27

Dhairya Ajitkumar Patel – 20906076

Krishna Kanth Mutta – 20919166

Yash Tiwari – 20911298

Question 1:

Part 1: What is the goal of sensor management in WSN?

Answer: Sensors are distributed randomly in a sensor field or a wide area to record the data or events that occurred in their surroundings and provide information about that event to the destination node. These sensors adopt a multihop path to transmit data from one location to the other. Hence, sensors, apart from transmitting the data, should also be capable to route the data to other sensors. Sensors should provide the data as well as route the data. Many sensors are present in the field, so the role of sensor management is to assign roles to each sensor so that the sensor nodes that are not needed can go into sleep mode and thus save energy which will help increase the life cycle of the network. Sensor management is done in the application layer of the protocol stack by the Sensor Management Protocol (SMP). Sensor management protocol performs the following tasks:

- Topology control – choosing which nodes will act as routers of data.
- Sensing mode selection – choosing which sensors will sense the data to improve QoS.
- Power consumption control – Ensuring that sensors that are not in use should go to sleep to save energy consumption.
- Fault tolerance.
- Security - detecting and prevention of intrusions.

References:

https://link.springer.com/chapter/10.1007/978-1-4020-7884-2_16

Part 2: Summarization of paper "Sensor Management using Relevance Feedback Learning"

Answer: This paper describes the management of sensors that are agile using the ML algorithm of Relevance Feedback learning.

In the first part of the paper, the author summarises several research works done on sensor management and scheduling such as multi-arm bandit formulation involving hidden Markov models, an optimal algorithm that tracks multiple targets with ESA containing steerable beam, a rollout algorithm that is based on the heuristics to approximate the stochastic dynamic programming, formulating a problem which classifies a large number of stationary objects with a multi-mode sensor. He also summarised that now divergence methods are the alternatives to sensor management because the sensing action maximizes the expected gain in information.

Divergence-based adaptivity measures such as Kullback-Leiber (KL) divergence are common learning metrics which is an iterative technique in which the system provides a set of items as a query to the user. Based on the user's response, the system adaptively chooses new queries based on the feedback. The goal of this iterative approach is to 'learn' from the user. A well-known example of this kind of approach is seen in Content-Based Image Retrieval (CIBR) which associates a probability of a correct image when compared with an image present in the database.

A similar type of management of sensors is done using the CIBR approach. The goal is to learn the number of the moving targets and their state occupying the surveillance region. The target tracking algorithm first calculates the joint multi-target probability density for the set of targets under surveillance in a recursive manner. The divergence matrix is used to determine the best query at each iteration of the method. Between the current joint multitarget probability density and the joint multitarget probability density after measurement, the sensor sensing decision will optimize the information gain. The system receives feedback on the current position of the targets, which is then utilized to update the probability density on the number and condition of targets.

In the paper comparison between two approaches that have feedback is carried out. The first is a particle filter (PF) based multi-target tracking algorithm and the second is a reinforcement learning approach for sensor management based on Renyi's divergence.

In the PF-based tracking algorithm, each particle is a sample of the JMPD and is an estimate of the number of the moving targets and their states. In the reinforcement-based algorithm, the drop in entropy caused by the measurement is a great indicator of sensing quality. When the sensor is ready for measurement, divergence is used to determine the best sensing action by enumerating all possible sensing actions, calculating the expected information gain for each, and updating the JMPD, which is then used to refine the multitarget density estimate for the next measurement.

The simulation was carried out to implement the algorithm and it showed that when the Renyi divergence parameter value is set to 0.5 it performs best and does not lose track of the 10 targets during 50 simulation runs. The author also performed comparisons between several divergence-based sensor management strategies and found out that the α-divergence (Renyi divergence) outperforms all the other strategies.

Other results in the case of tracking show that myopic sensor scheduling shows a loss of 22% of the time as compared to non-myopic sensor scheduling which has a loss of only 11% of the time. Hence, when the patterns are known, non-myopic are generally more preferred.

Part 3: Explain and compare, in your own words, the main attributes of the MQTT and COAP protocols.

Answer:

Constrained Application Protocol (COAP): In COAP protocol, packets are shared between different nodes which are controlled by the server of COAP. It has no acknowledgment of the messages.

Message Query Telemetry Transport (MQTT): This protocol is generally used in IoT devices. In this protocol, the client receives the information through the broker. This broker is the mediator who then makes segmentation the message into labels and then it is sent to the client.

- The key differences between COAP and MQTT protocols are as follows:
- The COAP protocol is client-server-based while the MQTT protocol is communication-based.
- The communication type of COAP is based on the Request-Response model while for MQTT, it is based on the Publish-Subscribe model.
- COAP uses both synchronous and asynchronous messaging models. However, MQTT used only an asynchronous messaging model.
- COAP uses UDP protocol in the transport layer, while MQTT mainly uses Transmission control protocol.
- The header size for COAP is 4 bytes while MQTT has only 2 bytes.
- Message labeling is not possible in MQTT while it is a feature in COAP protocol.
- COAP is mainly used in Utility area networks while MQTT is used generally in IoT applications.

References:

Lecture slides on COAP and MQTT protocols

https://www.geeksforgeeks.org/difference-between-coap-and-mqtt-protocols/

Question 2: Cognitive Sensor Networks

Part 1: What is cognitive sensor networks?

Answer: A new kind of technology known as cognitive technology is necessary for self-managing networks. They ought to comprehend what the application is attempting to achieve, and the application itself ought to be capable of comprehending what the network is capable of at any given time. By learning the application needs and selecting the network protocol that will satisfy them on the fly, this would enable a network to utilise new capabilities. The concept of spectrum allocation and usage is being approached in a fundamentally new way by cognitive technology. An intelligent wireless communication system known as a Cognitive Radio (CR) adjusts its internal parameters in response to its environment to provide a stable and effective transmission. Using CR technology, unlicensed (secondary) users can regularly scan the spectrum for open (vacant) channels and utilise them even though the licenced (primary) users initially control them. A cognitive sensor network is made up of numerous small, inexpensive sensors, each of which uses a finite amount of battery power. There would be an additional state called sensing state in a cognitive sensor network where the sensor nodes would sense the spectrum to look for spectrum opportunities or spectrum gaps. The primary benefit of this technology is its spectrum, which may be used to transmit data using underlay or overlay techniques.

Part 2: What makes them different from traditional sensor networks?

Answer: The creation of distributed networks with compact and affordable communication nodes has been made possible by the quick advancements in computing power, memory size, and radio technology. In comparison to a conventional wired sensor system, these nodes can sense, communicate, and be deployed for a lot less money which are called as Wireless sensor networks (WSNs), also the traditional sensor networks. The restricted bandwidth allotted to traditional networks, which is often the industrial, scientific, and medical (ISM) band, is just one of the difficulties they face. Spectrum shortage is a significant issue with these traditional networks. A revolutionary concept of Cognitive sensor networks has been presented to address this issue of spectrum scarcity in the traditional sensor networks. Importantly, in Cognitive sensor networks an additional operation of Spectrum sensing (SS) which is useful for periodically observing a frequency band to identify the presence or absence of primary users.

Part 3: Why are we interested in them?

Answer: Cognitive wireless sensor networks have many advantages which make us more inclined towards these devices. Few of the advantages are mentioned below –

1. Efficient spectrum utilization: Currently we have increased level of interference from various wireless systems (WSNs). Dynamic spectrum access in cognitive radio network is able to make the secondary users cooperate with other types of users and which leads to efficient spectrum utilization.
2. Multiple channels utilization: In cognitive wireless sensor networks, multiple channels are accessed which would discard potential challenges like packet collisions or packet losses which might decrease the communication reliability and increase the power consumption.
3. Energy efficiency: Cognitive wireless sensor networks would be able to change their operating parameters based on the surrounding channel conditions which would eliminate or decrease the packet losses that happens in the traditional wireless sensor networks.

4. Global operability: In cognitive wireless sensor networks, the sensor nodes with cognitive capabilities would avoid the issue of unavailability of certain bands due to various spectrum regulations. This enhances the global operability.

Part 4: Briefly discuss challenges one faces in trying to realize cognition in IOT and WSN.

Answer: Traditional wireless sensor networks faced numerous practical challenges as a result of the process of defending the rights of primary users in cognitive sensor networks. Some of these challenges in realizing cognition are mentioned below –

1. Miss-Detection and False Alarm Probability: False alarms prevent the effective use of the spectrum that is available, and miss detections leads to interference with primary users. Furthermore, low throughput, frequent frequency switching, and high delay are caused by false alarms and miss detection. The miss-detection probability is a measure that recognizes the presence of the primary signal on the channel, while false-alarm probability is a measure to identify the sensor node that fails to identify the non-existence of the primary signal.

2. Installation cost: In cognitive wireless sensor networks (CWSN), installation cost incurred would be high as many sensor nodes are deployed in the in the CWSN to cover the required region. Also, there are many other additional components that would be required which would increase the installation cost of the system.

3. Power Consumption: Cognitive radio wireless sensors would need power for handling frequent spectrum handoff. This action of spectrum handoff is apart from the regular tasks involved in wireless sensor networks. The primary user in the cognitive radio network needs continuous monitoring and it needs numerous antennas which would lead to high power utilization.

4. Security: The CR wireless sensors are vulnerable to attacks, and their data may also be stolen. The lack of collaboration between SU and PU communication makes CWSN more vulnerable to security threats than traditional WSN. The information gathered by CR wireless sensors may be destroyed, altered, or sniffed by unauthorized parties.

5. Hardware: The cognitive ability of the cognitive radio unit necessitates adaptation to the transmission variables, such as modulation, transmission power, and carrier frequency. The cognitive radio unit needs to handle spectrum movement, select the best channel, and other tasks. Data transmission and reception are the responsibilities of the communication unit. As a result, designing intelligent hardware for the Cognitive wireless sensor network is a challenging procedure that requires extensive hardware installation.

Part 5: Explain what is meant by spectrum sensing.

Answer: The process of routinely scanning a certain frequency band for the presence or absence of principal users is known as spectrum sensing. In cognitive sensor networks, spectrum sensing is a crucial step in learning radio environment. The fundamental goal of a cognitive radio is to facilitate spectrum reuse or spectrum sharing, which enables secondary networks and users to use the primary users' (PUs') allocated or licensed spectrum for communication when those users are not completely utilizing it. To do this, secondary users (SUs) must often conduct spectrum sensing, which involves identifying the existence of the primary users. When the primary users are active, the secondary users must identify their presence

with a high probability and either leave the channel immediately or reduce transmit power within some time. There are two types of spectrum sensing methods: Local Spectrum Sensing and Cooperative Spectrum Sensing.

Part 6: Identify and three spectrum sensing techniques. Summarize each one of them briefly. If you are to combine two of them in one implementation to achieve improved performance, which two of the three you have identified would you choose? Justify why.

Answer: Some of the spectrum sensing techniques are outlined below –

1. Energy Detection (ED): Based on the detected energy, the principal signal is found using a non-coherent detection approach. Energy detection (ED) is the most often used sensing method in cooperative sensing because it is straightforward and does not require a priori knowledge of the principal user signal. But ED suffers from the noise uncertainty and requires a large number of samples for achieving a high probability of detection.
2. Cyclostationary Feature detection (CFD): It makes use of the primary signal's periodicity to determine the existence of primary users (PU). These cyclostationary signals have periodic statistics and spectral correlation characteristics because of their periodicity, which are absent in stationary noise and interference. Therefore, cyclostationary feature detection outperforms energy detection in low SNR regions and is robust to noise uncertainty. High computing complexity and longer sensing time are drawbacks of this method. These problems make this detection technique less popular in cooperative sensing than energy detection.
3. Eigenvalue based sensing (EBS): Eigenvalue-based spectrum sensing does not require much prior knowledge about the primary user signals and noise power. However, this method's drawbacks include its high operating complexity and prolonged sensing time.

Of the 3 techniques outlined, for achieving a better performance, we can combine Energy Detection (ED) and Eigenvalue based sensing techniques (EBS). Here, as stated above, ED suffers from noise uncertainty which could be overcome by EBS. Moreover, EBS has high operating complexity and prolonged sensing time which would not be a right approach to use this technique individually. So, we can use EBS at situations where ED couldn't accurately detect the presence of primary users.

References:

- Lecture Slides
- Kockaya, K., Develi, I. Spectrum sensing in cognitive radio networks: threshold optimization and analysis. J Wireless Com Network 2020, 255 (2020). https://doi.org/10.1186/s13638-020-01870-7

Question 3: Time Synchronization

(a) Refer to TDOA localization in a three-dimensional space. Assume that five reference nodes are known at Node 1: (0,3,0), Node 2: (6,0,0), Node 3: (3,4,0), Node 4: (4, 3, 0), and Node 5: (0, 0, 8) respectively. Also, for node U, we know the following propagation times: U11 = 0s, U12 = 1s, U13 = 0.7s, U14 = 0.7s, & U15 = 1.7s. The velocity of propagation is v.

(i) Find the unknown location (xU , yU , zU).

(ii) Now assume that the propagation speed is known to be 8.7 m/s. Find the unknown location (xU , yU , zU).

In [1]:
```
import numpy as np
```

Question 3A

part i

In [6]:
```
A = np.array([[0,3,0,0,0],[6,0,0,-1,0.5],[3,4,0,-0.7,0.245],[4,3,0,-0.7,0.245],[0,0,
B = np.array([4.5,18,12.5,12.5,32])
```

In [10]:
```
At = np.transpose(A)
```

In [14]:
```
w = np.linalg.inv(At.dot(A)).dot(At).dot(B)
```

In [15]:
```
w
```

Out[15]:
```
array([ 1.5       ,  1.5       , -0.95833333, 11.47619048, 40.95238095])
```

part ii

In [25]:
```
A = np.array([[0,3,0,0],[6,0,0,-1],[3,4,0,-0.7],[4,3,0,-0.7],[0,0,8,-1.7]])
B = np.array([4.5,18,12.5,12.5,32])
D = np.array([0,-0.5,-0.245,-0.245,-1.445])
```

In [26]:
```
At = np.transpose(A)
```

In [27]:
```
M = np.linalg.inv(At.dot(A)).dot(At).dot(B)
```

In [28]:
```
N =  np.linalg.inv(At.dot(A)).dot(At).dot(D)
```

In [29]:
```
w = M  + N*75.69
```

In [30]:
```
w
```

Out[30]:
```
array([ 0.10207235,  1.8994079 , -5.51526841, 19.55876633])
```

(a) Given,

$(x_1, y_1, z_1) = (0, 3, 0)$ $(x_2, y_2, z_2) = (6, 0, 0)$

$(x_3, y_3, z_3) = (3, 4, 0)$ $(x_4, y_4, z_4) = (4, 3, 0)$

$(x_5, y_5, z_5) = (0, 0, 8)$

$t_{11} = 0$ $t_{12} = 1$ $t_{13} = 0.7$ $t_{14} = 0.7$ $t_{15} = 1.7$

Soln:

(i) $Aw = b$

$\Rightarrow w = (A^T A)^T A^T \times b$

where

$$A = \begin{bmatrix} x_1 & y_1 & z_1 & -t_{11} & t_{11}^2/2 \\ x_2 & y_2 & z_2 & -t_{12} & t_{12}^2/2 \\ x_3 & y_3 & z_3 & -t_{13} & t_{13}^2/2 \\ x_4 & y_4 & z_4 & -t_{14} & t_{14}^2/2 \\ x_5 & y_5 & z_5 & -t_{15} & t_{15}^2/2 \end{bmatrix} \quad b = \begin{bmatrix} r_1^2/2 \\ r_2^2/2 \\ r_3^2/2 \\ r_4^2/2 \\ r_5^2/2 \end{bmatrix}$$

$$w = \begin{bmatrix} x_v \\ y_v \\ z_v \\ v^3 \\ v^2 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 3 & 0 & 0 & 0 \\ 6 & 0 & 0 & -1 & +0.5 \\ 3 & 4 & 0 & -0.7 & 0.245 \\ 4 & 3 & 0 & -0.7 & 0.245 \\ 0 & 0 & 8 & -1.7 & 1.445 \end{bmatrix}$$

$r_1^2 = x_1^2 + y_1^2 + z_1^2 = 0 + 9 + 0 = 9$

$r_2^2 = x_2^2 + y_2^2 + z_2^2 = 36 + 0 + 0 = 36$

$r_3^2 = x_3^2 + y_3^2 + z_3^2 = 16 + 9 + 0 = 25$

$r_4^2 = x_4^2 + y_4^2 + z_4^2 = 16 + 9 + 0 = 25$

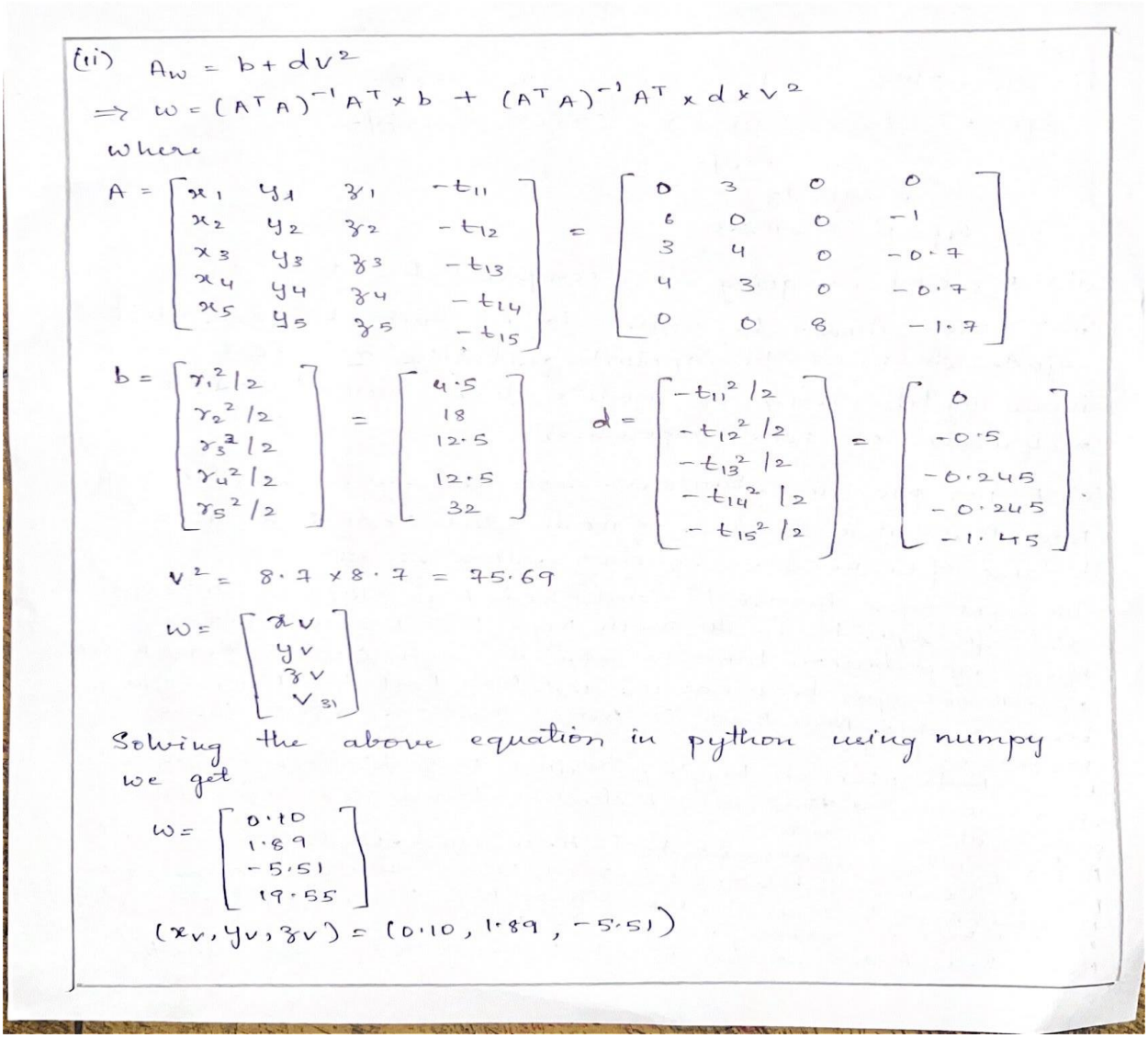
$r_5^2 = x_5^2 + y_5^2 + z_5^2 = 0 + 0 + 64 = 64$

$$\Rightarrow b = \begin{bmatrix} 4.5 \\ 18 \\ 12.5 \\ 12.5 \\ 32 \end{bmatrix}$$

The equation $w = (A^T A)^{-1} A^T b$ is solved in python by using numpy library

$$\Rightarrow w = \begin{bmatrix} 1.5 \\ 1.5 \\ -0.95 \\ 11.47 \\ 40.95 \end{bmatrix}$$

$(x_v, y_v, z_v) = (1.5, 1.5, -0.95)$



(b) Node A sends a synchronization request to node B at 3150 (on node A's clock). At 3250, node A receives the reply from node B with a time stamp of 3120.

(i) What is node A's clock offset with respect to the time at node B (you can ignore any processing delays at either node)?

(ii) Is node A's clock going too slow or too fast?

(iii) How should node A adjust its clock?

(c) Two nodes A and B use RBS to receive periodic acoustic synchronization signals from a reference node. Node A's clock shows 10 s when it receives the last synchronization beacon, while node B's clock shows 15 s. Node A detects an event at time 15 s, while node B detects the same event at time 19.5 s. Assume that node A is 100 m away from the synchronization source and node B is 400 m away from the synchronization source. Which node detected the event sooner and by how much? Assume a signal speed of 300 m/s.

(b)

(i) $T_1 = 3150$ $T_2 = 3120$ $T_3 = 3250$

offset = $(3120 - 3150) - (3250 - 3120)/2$

$= -30 - 130/2$

$= -160/2$

offset = -80 sec

(ii) A's clock is going slow compared to B.

(iii) 80 sec should be added to A's clock. However clock adjustments must be applied gradually as just increasing/decreasing a node's clock would lead to malfunction of node's operation.

(c) As per problem statement, acoustic signal speed is 300m/s Node A is at a distance of 100m and Node B is at a distance of 400m from synchronization source.

Time for sync beacon to reach Node A is $100/300 = 0.3333$ s.

Time for sync beacon to reach Node B is $400/300 = 1.3333$ s.

Hence, sync beacon takes 1s more to reach Node B than Node A.

Node A got sync beacon at 10s and Node B at 15s. When sync beacon reached Node B at 15s, Node A's clock would have progressed to 11s.

Hence Node A's clock lagging behind Node B's clock by 4s. Here we are assuming that Node A clock and Node B's clock has progress at same rate.

As per problem statement, Node A received event at 15s and Node B received the same event at 19.5s. In other words, from the time Node B received sync beacon, Node A received event after 4s and Node B received event after 4.5s.

Hence Node A detected the event 500ms sooner than Node B.

(d) TOA requires that all the reference nodes and the receiver have precise synchronized clocks and the transmitted signals be labeled with time stamps. TDOA measurements remove the requirement of an accurate clock at the receiver. Assume that five reference nodes have known positions (0, 0),(−1, −1),(0, 1),(3, 1), and (1, 4) respectively. We choose (0, 0) as the reference sensor for differential time-delays which are defined as t1r = t1 − tr = rs1 − rs2 v where v is the velocity of propagation, rsi is the distance between the unknown node and the i th node. Further assume that t12 = −1.4s, t13 = 0.4s, t14 = −1.6s, and t15 = −2.6s

(a) Find the unknown location (xt, yt).

(b) Now assume that the propagation speed is known as 1.8 m/s. Find the unknown location (xt, yt).

Question 3d

part i

In [34]:
```
A = np.array([[-1,-1,1.4,0.98],[0,1,-0.4,0.08],[3,1,1.6,1.28],[1,4,2.6,3.38]])
B = np.array([1,0.5,5,8.5])
```

In [35]:
```
At = np.transpose(A)
```

In [36]:
```
w = np.linalg.inv(At.dot(A)).dot(At).dot(B)
```

In [37]:
```
w
```

Out[37]:
```
array([ 0.875 , -0.125 , -0.9375,  3.125 ])
```

part ii

In [38]:
```
A = np.array([[-1,-1,1.4],[0,1,-0.4],[3,1,1.6],[1,4,2.6]])
B = np.array([1,0.5,5,8.5])
D = np.array([-0.98,-0.08,-1.28,-3.38])
```

In [39]:
```
At = np.transpose(A)
```

In [40]:
```
M = np.linalg.inv(At.dot(A)).dot(At).dot(B)
```

In [41]:
```
N =  np.linalg.inv(At.dot(A)).dot(At).dot(D)
```

In [42]:
```
w = M  + N*3.24
```

In [43]:
```
w
```

Out[43]:
```
array([ 0.89046008, -0.16447354, -1.0328121 ])
```

In [ ]:
```

```

(d)

(i) $Aw = b$

$w = (A^T A)^{-1} A^T \times B$

$$A = \begin{bmatrix} x_1 & y_1 & -t_{11} & t_{11}^2/2 \\ x_2 & y_2 & -t_{12} & t_{12}^2/2 \\ x_3 & y_3 & -t_{13} & t_{13}^2/2 \\ x_4 & y_4 & -t_{14} & t_{14}^2/2 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1.4 & 0.98 \\ 0 & .1 & -0.4 & 0.08 \\ 3 & 1 & 1.6 & 1.28 \\ 1 & 4 & 2.6 & 3.38 \end{bmatrix}$$

$$B = \frac{1}{2}\begin{bmatrix} 2 \\ 1 \\ 10 \\ 17 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \\ 5 \\ 8.5 \end{bmatrix} \qquad w = \begin{bmatrix} x_v \\ y_v \\ v_{31} \\ v^2 \end{bmatrix}$$

Solving the above equation in python we get

$$w = \begin{bmatrix} 0.875 \\ -0.125 \\ -0.9375 \\ 3.125 \end{bmatrix}$$

$v = (0.875, -0.125)$

(ii) $Aw = b + dv^2$

$w = (A^T A)^{-1} A^T \times b + (A^T A^{-1}) \; A^T \times d \times v^2$

$v^2 = (1.8 \times 1.8) = 3.24$

$$A = \begin{bmatrix} x_1 & y_1 & -t_{11} \\ x_2 & y_2 & -t_{12} \\ x_3 & y_3 & -t_{13} \\ x_4 & y_4 & -t_{14} \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1.4 \\ 0 & 3 & -0.4 \\ 3 & 1 & 1.6 \\ 1 & 4 & 2.6 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ 0.5 \\ 5 \\ 8.5 \end{bmatrix} = \begin{bmatrix} r_1^2/2 \\ r_2^2/2 \\ r_3^2/2 \\ r_4^2/2 \end{bmatrix} \qquad w = \begin{bmatrix} x_v \\ y_v \\ v_{31} \end{bmatrix}$$

$$d = \begin{bmatrix} -t_{11}^2/2 \\ -t_{12}^2/2 \\ -t_{13}^2/2 \\ -t_{14}^2/2 \end{bmatrix} = \begin{bmatrix} -0.98 \\ -0.08 \\ -1.28 \\ -3.38 \end{bmatrix}$$

Solving the above equation in python we get

$$w = \begin{bmatrix} 0.890 \\ -0.164 \\ -1.032 \end{bmatrix}$$

$$V = (0.89, -0.16)$$

Question 5: Coverage Control

2. Topology for Coverage

Part a: What is the difference between the detection optimal coverage and estimation optimal coverage?

Answer: Differences between detection optimal coverage and estimation optimal coverage are –

S.No.	Detection Optimal coverage	Estimation Optimal coverage
1.	The main application of Detection optimal coverage is the detection of an event that occurred at a particular location.	The main application of Estimation optimal coverage is to estimate the signal parameters.
2.	For detection optimal coverage, the sensing quality of a sensor can be represented by its detection probability.	For estimation optimal coverage, the sensing quality of a sensor can be represented by its estimation error.
3.	Higher the detection probability value, better the chances of sensor covering a given point.	Smaller the estimation error, better the chances of estimated parameter being more accurate.
4.	In the case of a decreasing function, detection probability decreases when distance decreases.	In the case of a decreasing function, the probability of estimation error decreases when distance increases.
5.	Example: Majority voting	Example: Best linear unbiased estimator (BLUE)

Part b: Performing Simulated Annealing in a field of 500x500m area with 17 targets

Answer:

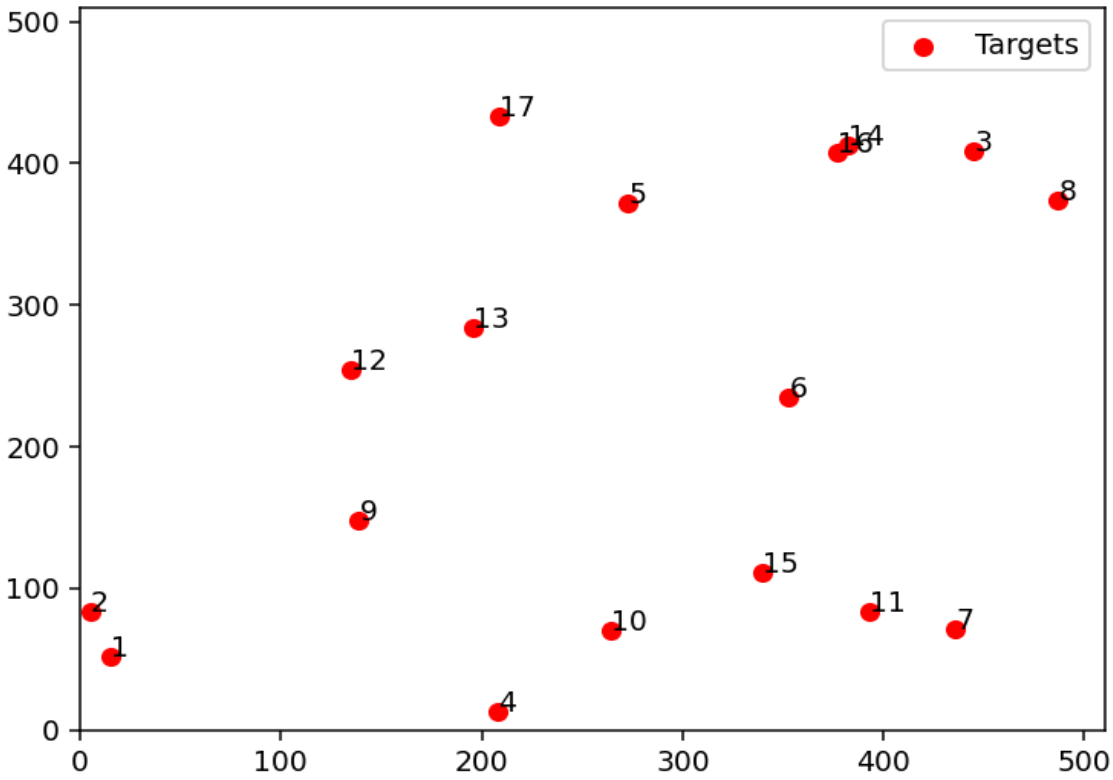
Data and Assumptions:

Deliverable 1: Device a function to distribute the 17 targets around the surveillance area.

We have 17 targets to be placed randomly in the field of 500x500m area. We have chosen 1x1 boxes on a 500x500 field to accumulate the 17 targets. Hence, we get 250,000 possible locations for the placement of possible 17 targets in the field.

We also have a constraint that a particular same location should not contain two targets, this constraint is taken care in the code as mentioned below

The snapshot of the function and the plot of the field having 17 targets are shown as follows:



```python
# Reference of SA algorithm is taken from http://apmonitor.com/me575/index.php/Main/SimulatedAnnealing
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import random
import math

# Creating a field of targets randomly spread on 500x500m area. The target positions change with each run
# The below function returns the position / coordinates of targets on the field
def target_sites():
    X = 500
    Y = 500
    targets = 17
    tot_targets = 250000 # (500x500) possible locations to choose from
    x_coor = []; y_coor = []
    count=0
    while count < 17:
        a = random.randint(1, 500)
        if a not in x_coor: #constraints that are taken care of
            x_coor.append(a)
            count = count+1

    count = 0
    while count < 17:
        b = random.randint(1, 500)
        y_coor.append(b)
        count = count+1
    print("X_coor are:::::::: ", x_coor)
    print("Y_coor are:::::::: ", y_coor)
    return x_coor, y_coor
```

Deliverable 2: Device simulated annealing algorithm to compute an optimal node placement, using the model proposed above.

To perform simulated annealing, we are taking the following data and assumptions:

First, we have no restrictions on choosing the number of types of sensors S1, S2, and S3. The cost of sensors is 300, 170, and 65 units respectively and the range of the sensors is 100, 70, and 30 meters respectively.

Also, the function coverage checker ensures constraint 1 is resolved.
Constraint 2 is taken care of in the code, which was each target and sensor should have a different location.

The cost function is defined as:
Cost Function: (s1 * c1) + (s2 * c2) + (s3 * c3), where s1, s2, s3 are the number of sensors and c1, c2, c3 are the costs of each type of sensor.

For performing simulated annealing with k=1 and k =2, we have chosen a random number of sensors between 1 to 25 and for k=3, as we require a greater number of sensors, we have increased the range to 50.

In our simulation, for a fixed value of k, there is a fixed location of target sites. If the value of k changes, then the location of target sites also changes.

The number of cycles is also kept as a parameter, that can be changed by the user. For k=1 and k=2 we have kept the cycles at 150 while for k=3 as we need the more optimum location of sensors w.r.t. costs, we are keeping the cycles to 1500. For both scenarios, we have kept the number of iterations to 50.

The initial temp is taken as 2.8036 while the final temp is chosen as 0.1447. After 50 iterations the temp is lowered by 0.9413 value. This is taken from the reference: (http://apmonitor.com/me575/index.php/Main/SimulatedAnnealing)

During each iteration we did the following steps:

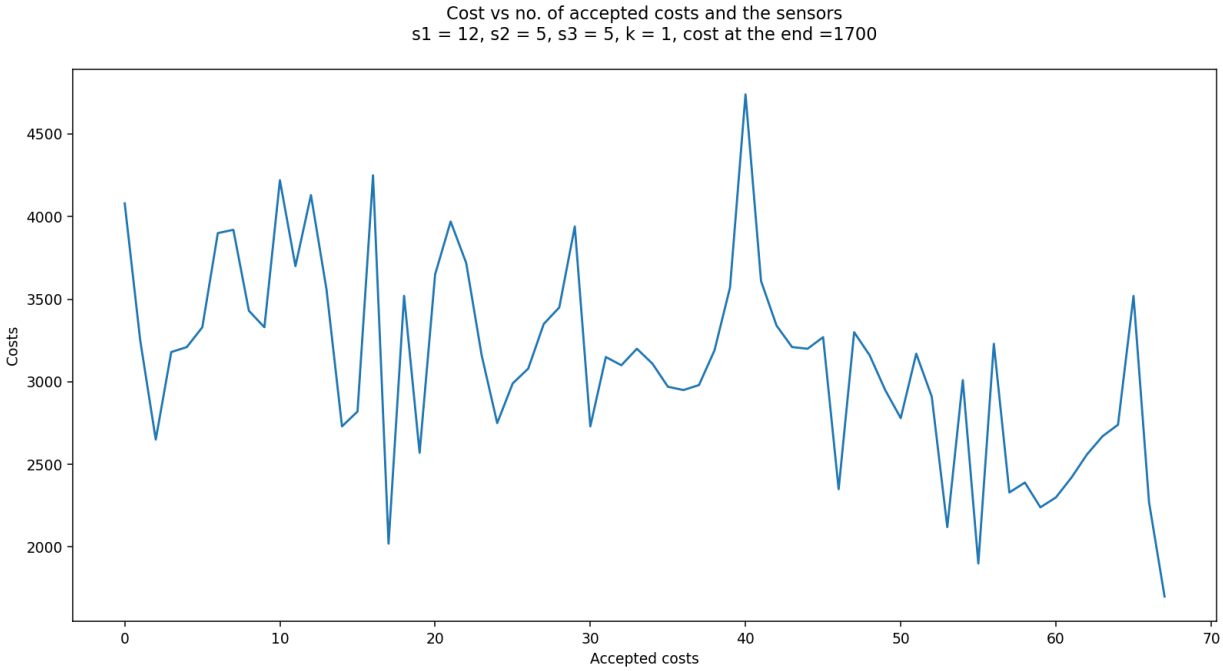
1. Randomly distribute the randomly chosen sensors of each type in the grid. During each iteration choose number of S1, S2 & S3 sensors randomly.
2. Check if each target was covered by the 'k' number of sensors (via **Check_coverage** python method). 'k' was taken as 1, 2, and 3 in our case.
3. If coverage was false, we again went back to step 1.
4. If the coverage was true, the cost was computed for that arrangement of sensors.
5. If the new cost was better than the earlier cost, it was momentarily considered the best cost, and the sensor configuration was saved.
6. If the new cost was worse than the earlier cost, a predefined probability function was used to determine if the worse cost could be accepted or not.

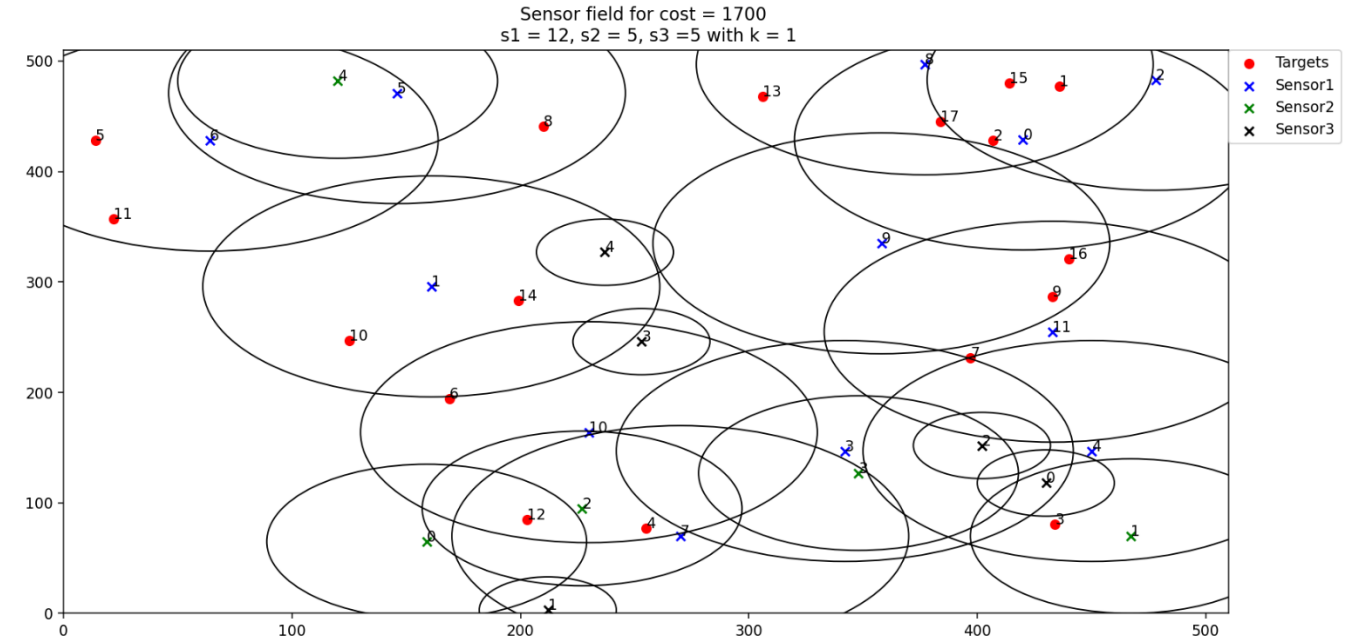
We generated a random number between 0 and 1. If the random number > probability the worst cost is not accepted. If the random number < probability, then the worst cost is accepted.

Simulation results:

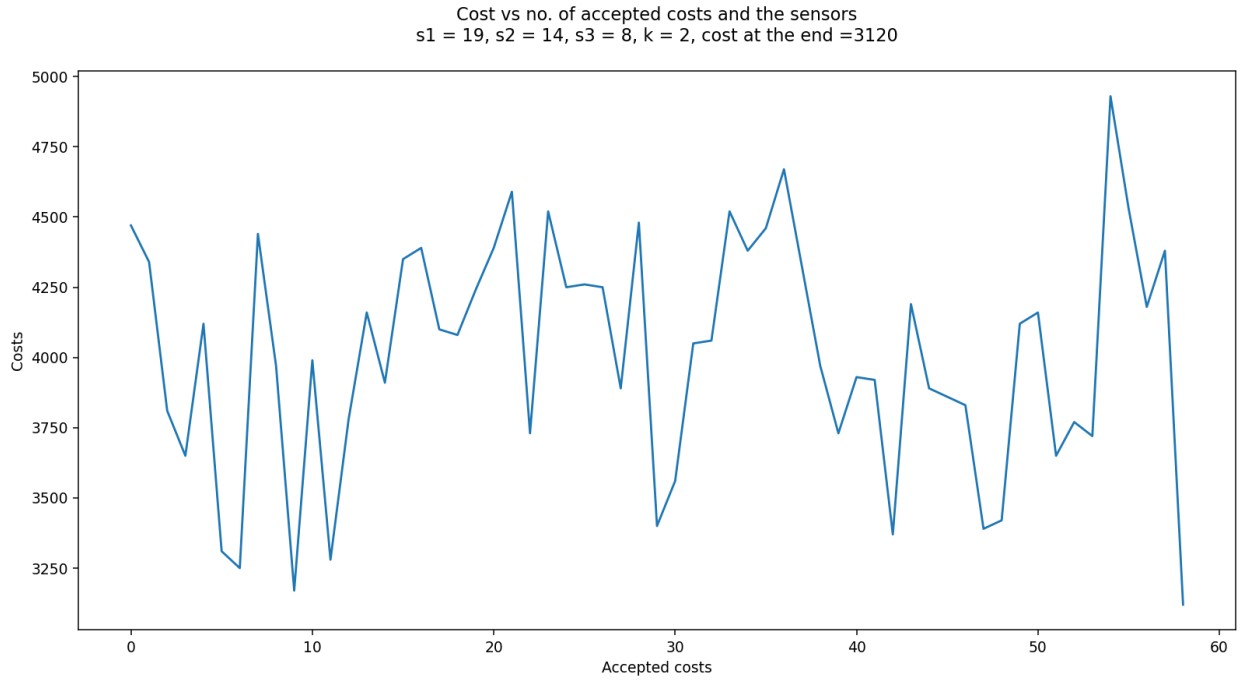
For k = 1,

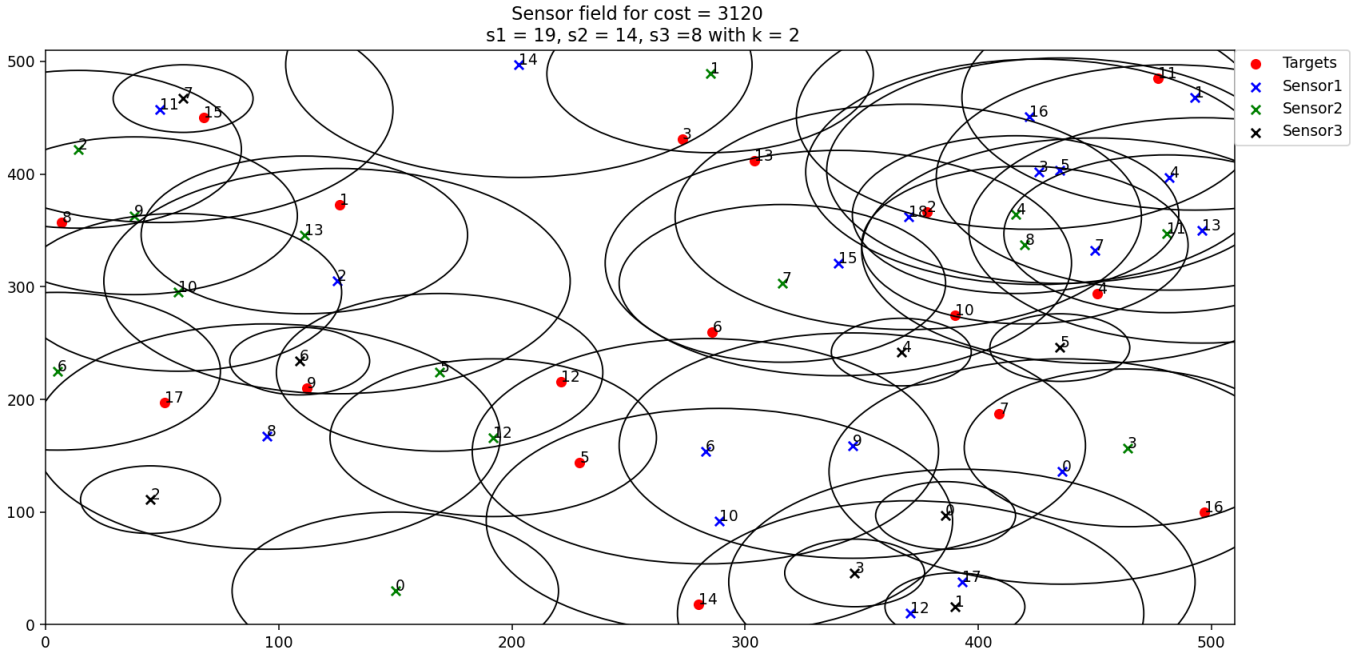
S1 = 12, S2 = 5, S3 = 5 and we get cost = 1700



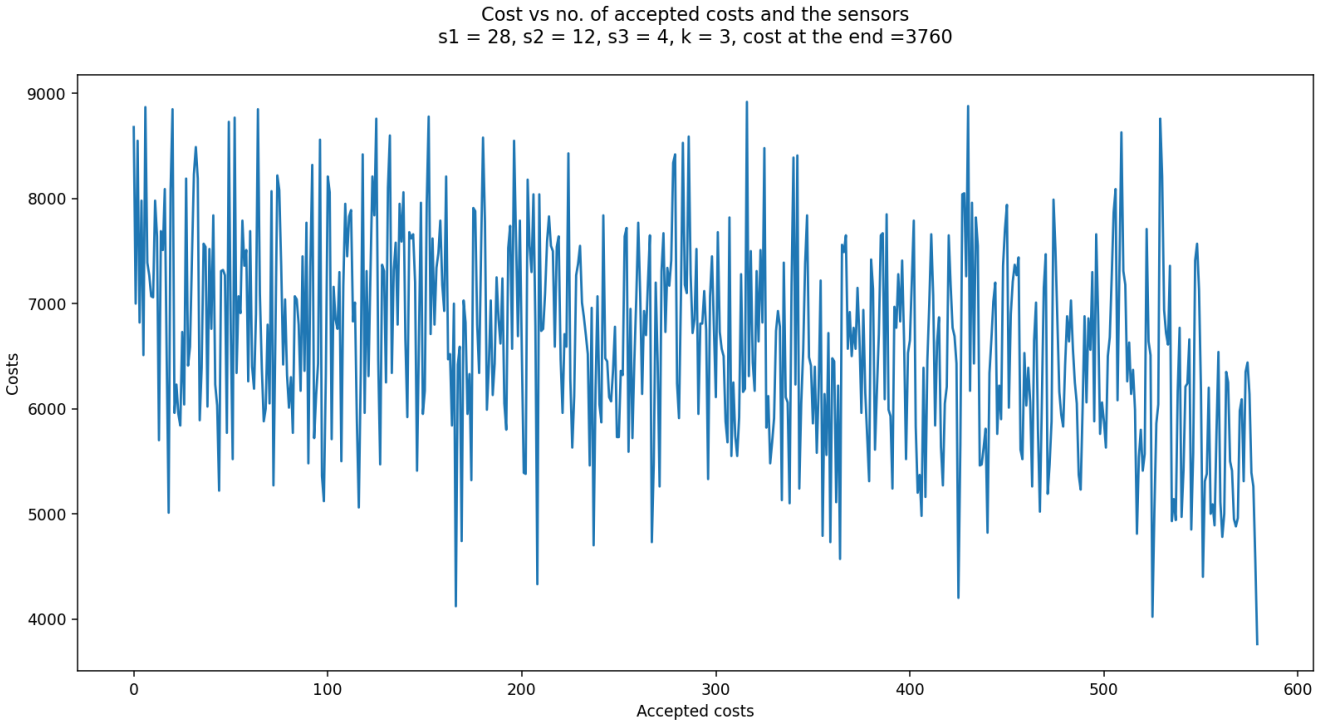


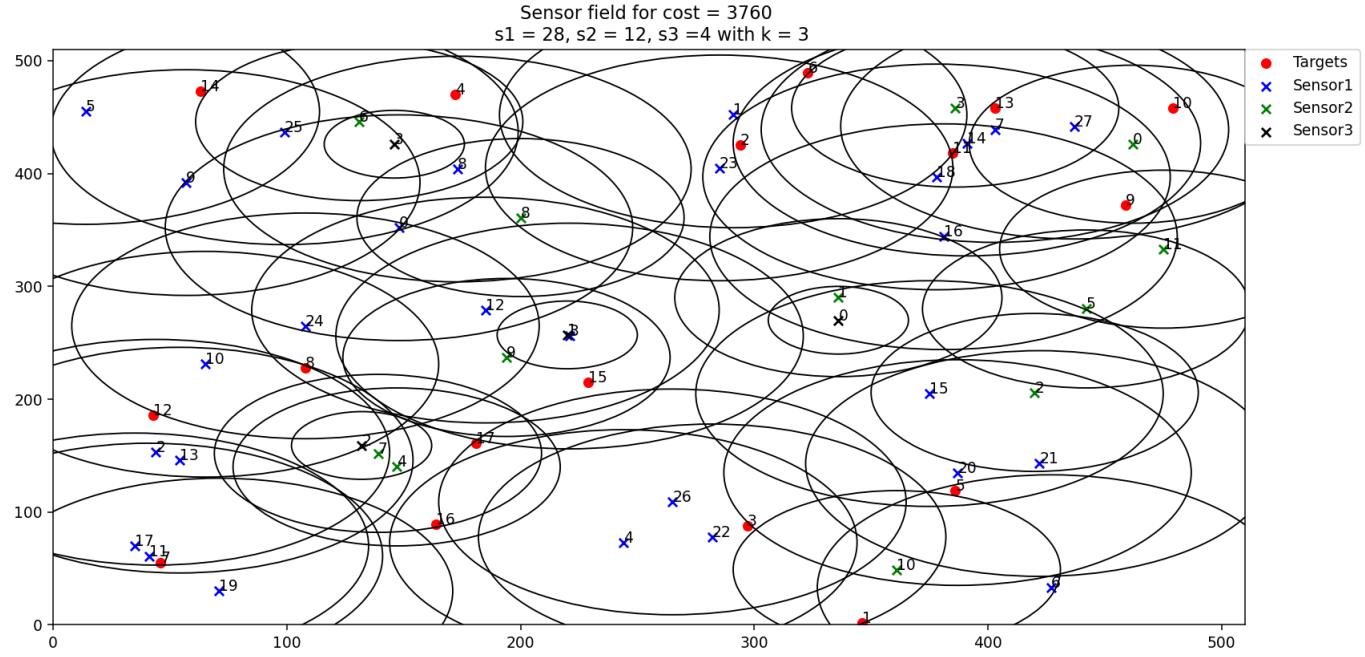
For k = 2,
S1 = 19, S2 = 14, S3 = 8 and we get cost = 3120





For k = 3,
S1 = 28, S2 = 12, S3 = 4 and we get cost = 3760
Here the cycles are taken as 1500 as we wanted to have a lower value of cost.





The python code is as follows:

A2_1_2.py

```python
# Reference of SA algorithm is taken from http://apmonitor.com/me575/index.php/Main/SimulatedAnnealing
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import random
import math

# Creating a field of targets randomly spread on 500x500m area. The target positions change with each run
# The below function returns the position / coordinates of targets on the field
def target_sites():
    X = 500
    Y = 500
    targets = 17
    tot_targets = 250000 # (500x500) possible locations to choose from
    x_coor = []; y_coor = []
    count=0
    while count < 17:
        a = random.randint(1, 500)
        if a not in x_coor: #constraints that are taken care of
            x_coor.append(a)
            count = count+1

    count = 0
    while count < 17:
        b = random.randint(1, 500)
        y_coor.append(b)
        count = count+1
    print("X_coor are:::::::: ", x_coor)
    print("Y_coor are:::::::: ", y_coor)
    return x_coor, y_coor
```

A2_1_2.py > target_sites

```python
#placing sensor on the field such that all the constraints are satisfied
#The below function returns the position / coordinates of all the 3 sensors
def sen_loc(x_coor,y_coor):
    s1_coor_x = []; s1_coor_y = []
    s2_coor_x = []; s2_coor_y = []
    s3_coor_x = []; s3_coor_y = []

    count = 0
    while count < s1:
        c = random.randint(1, 500)
        if c not in (x_coor and s1_coor_x): #constraints to be satisfied wrt assignment
            s1_coor_x.append(c)
            count = count+1
    count = 0
    while count < s1:
        d = random.randint(1, 500)
        s1_coor_y.append(d)
        count = count+1

    count = 0
    while count < s2:
        d = random.randint(1, 500)
        if d not in (x_coor and s1_coor_x and s2_coor_x):
            s2_coor_x.append(d)
            count = count+1
    count = 0
    while count < s2:
        e = random.randint(1, 500)
        s2_coor_y.append(e)
        count = count+1

```

A2_1_2.py > sen_loc

```
    count = 0
    while count < s3:
        f = random.randint(1, 500)
        if f not in (x_coor and s1_coor_x and s2_coor_x and s3_coor_x):
            s3_coor_x.append(f)
            count = count+1

    count = 0
    while count < s3:
        e = random.randint(1, 500)
        s3_coor_y.append(e)
        count = count+1

    return s1_coor_x, s1_coor_y, s2_coor_x, s2_coor_y, s3_coor_x, s3_coor_y

#plotting the sensor field location and their range
def plotting(s1_coor_x, s1_coor_y, s2_coor_x, s2_coor_y,s3_coor_x, s3_coor_y,x_coor,y_coor,s1,s2,s3):
    plt.scatter(x_coor, y_coor, color='red', label='Targets')
    for i, label in enumerate(range(1,18)):
        plt.annotate(label, (x_coor[i], y_coor[i]))
    plt.scatter(s1_coor_x, s1_coor_y, color='blue', label='Sensor1', marker = 'x')
    for i, label in enumerate(range(s1)):
        plt.annotate(label, (s1_coor_x[i], s1_coor_y[i]))
    plt.scatter(s2_coor_x, s2_coor_y, color='green', label='Sensor2', marker = 'x')
    for i, label in enumerate(range(s2)):
        plt.annotate(label, (s2_coor_x[i], s2_coor_y[i]))
    plt.scatter(s3_coor_x, s3_coor_y, color='black', label='Sensor3', marker = 'x')
    for i, label in enumerate(range(s3)):
        plt.annotate(label, (s3_coor_x[i], s3_coor_y[i]))
    for i in range(s1):
        draw_circle = plt.Circle((s1_coor_x[i],s1_coor_y[i]), 100, fill=False)
        plt.gcf().gca().add_artist(draw_circle)
```

A2_1_2.py > sen_loc

```
    for i in range(s1):
        draw_circle = plt.Circle((s1_coor_x[i],s1_coor_y[i]), 100, fill=False)
        plt.gcf().gca().add_artist(draw_circle)
    for i in range(s2):
        draw_circle = plt.Circle((s2_coor_x[i],s2_coor_y[i]), 70, fill=False)
        plt.gcf().gca().add_artist(draw_circle)
    for i in range(s3):
        draw_circle = plt.Circle((s3_coor_x[i],s3_coor_y[i]), 30, fill=False)
        plt.gcf().gca().add_artist(draw_circle)
    plt.xlim(0,510); plt.ylim(0,510)
    # plt.legend()
    plt.legend(bbox_to_anchor=(1, 1), loc=2, borderaxespad=0.)
    plt.draw(); plt.waitforbuttonpress(0); plt.close()
```

A2_1_2.py > ...

```python
#Checking coverage of all targets in the list and then returns True if all the target positions are k covered
def Check_coverage(k,s1,s2,s3):
    Is_coverage = 17*[0]
    #s1
    for i in range(17):
        for j in range(s1):
            dist = math.sqrt((x_coor[i] - s1_coor_x[j])**2 + (y_coor[i] - s1_coor_y[j])**2 ) #finding distance
            if dist <= r1:
                Is_coverage[i] =Is_coverage[i] + 1

    #s2
    for i in range(17):
        for j in range(s2):
            dist = math.sqrt((x_coor[i] - s2_coor_x[j])**2 + (y_coor[i] - s2_coor_y[j])**2 )
            if dist <= r2:
                Is_coverage[i] =Is_coverage[i] + 1

    #s3
    for i in range(17):
        for j in range(s3):
            dist = math.sqrt((x_coor[i] - s3_coor_x[j])**2 + (y_coor[i] - s3_coor_y[j])**2 )
            if dist <= r3:
                Is_coverage[i] =Is_coverage[i] + 1

    if k ==1:
        for i in range(17):
            if Is_coverage[i] <1:
                return False

    if k ==2:
        for i in range(17):
            if Is_coverage[i] <2:
                return False
```

A2_1_2.py > Check_coverage

```python
    if k ==3:
        for i in range(17):
            if Is_coverage[i] <3:
                return False
    print("IS_coverage", Is_coverage) #prints the list of length 17 with values showing the number of sensors that covers that particular target

    return True

s1 = 20; s2 = 20; s3 = 10 #initial number of sensors which changes in the following code wrt the cost
r1 = 100; r2 = 70; r3 = 30 #range of sensors
#cost of sensors
c1 = 300
c2 = 170
c3 = 65
cost = s1*r1 + s2*r2 + s3*r3 #initial cost
x_coor, y_coor = target_sites() #position of targets
initial_cost  = cost
iterations = 50 #each cycle has 50 iterations
cycles = 1500   #cycles which is a parameter that can be altered by the user
temp1 = -1.0/math.log(0.7); temp50 = -1.0/math.log(0.001) #inital temp and final temp
fraction = (temp50/temp1)**(1.0/(cycles-1.0))
temp = temp1
deltaAvg = 0.0
best_cost = initial_cost #choosing best cost as initial cost
na = 1 #no_of_acceptable_solutions
all_cost = [] #list to store all costs
all_approved_costs = [] #list to store all accepted cost
all_cost.append(cost)
```

A2_1_2.py > ...

```
#SIMULATED ANNEALING
for i in range(cycles+1):
    coverage = False
    for j in range(iterations):
        while(coverage == False):
            #choosing sensors randomly between 1 to 50
            s1 = random.randint(1,50)
            s2 = random.randint(1, 50)
            s3 = random.randint(1,25)

            s1_coor_x, s1_coor_y, s2_coor_x, s2_coor_y,s3_coor_x, s3_coor_y = sen_loc(x_coor, y_coor) #finding sensor locations
            k=3 #value of k
            coverage = Check_coverage(k,s1,s2,s3) #check if a target is k-connected
            #if targets are k connected find the cost, compare with best cost, if new cost is less than best cost replace the best cost with
            #new cost, else accept the worst cost based on the probablity function of acceptance
            if coverage==True:
                new_cost = s1*r1 + s2*r2 + s3*r3
                all_cost.append(new_cost)    #history
                print("Best cost: ", best_cost, " New cost: ", new_cost)
                if (best_cost > new_cost):
                    delta = abs(best_cost-new_cost)
                    best_cost = new_cost
                    #storing the best location of sensors for our plot
                    final_s1_x = s1_coor_x; final_s1_y = s1_coor_y
                    final_s2_x = s2_coor_x; final_s2_y = s2_coor_y
                    final_s3_x = s3_coor_x; final_s3_y = s3_coor_y
                    sen_1 = s1; sen_2 = s2; sen_3 = s3
                    all_approved_costs.append(best_cost)
                    na = na+1
                else:
                    # print("New cost is worse :(")
                    delta = abs(best_cost - new_cost)
```

A2_1_2.py > i

```
                    delta = abs(best_cost - new_cost)
                    if deltaAvg == 0:
                        deltaAvg = delta
                    probab_of_acceptance = math.exp(-delta/(deltaAvg * temp)) #probablity of acceptance of worst cost
                    random_num = random.uniform(0, 1) #generating a random number

                    if random_num > probab_of_acceptance: #if random_num > prob_of_acceptance, donot accept the worst solution
                        continue
                    else: #accept worst cost
                        #accept worst cost
                        best_cost = new_cost #replacing best cost with new cost

                        final_s1_x = s1_coor_x; final_s1_y = s1_coor_y
                        final_s2_x = s2_coor_x; final_s2_y = s2_coor_y
                        final_s3_x = s3_coor_x; final_s3_y = s3_coor_y
                        sen_1 = s1; sen_2 = s2; sen_3 = s3
                        all_approved_costs.append(best_cost)
                        na = na+1
    deltaAvg = (deltaAvg * (na-1.0) + delta) / (na)  #updating delta avg
    #lower temp
    temp = fraction* temp
print("All approved costs are: ", all_approved_costs)
plt.title("Cost vs no. of accepted costs and the sensors\n s1 = %d, s2 = %d, s3 = %d, k = %d, cost at the end =%d \n" %(sen_1, sen_2, sen_3, k,all_approved_
plt.xlabel("Accepted costs"); plt.ylabel("Costs")
plt.plot(all_approved_costs)
plt.draw()
plt.waitforbuttonpress(0)
plt.close()
print(len(all_approved_costs))
plt.title("Sensor field for cost = %d \n s1 = %d, s2 = %d, s3 =%d with k = %d" %(all_approved_costs[-1], sen_1, sen_2, sen_3, k))
plott = plotting(final_s1_x, final_s1_y, final_s2_x, final_s2_y, final_s3_x, final_s3_y, x_coor, y_coor,sen_1,sen_2,sen_3)
```