DISTRIBUTED DEPLOYMENT OF WIRELESS SENSOR NETWORKS

Thesis submitted in partial fulfilment of
the requirements for the degree of

Bachelor of Technology
in
Electronics and Electrical Communication Engineering

by

Jishnu Dey
(Roll No.: 12EC10026)

Under the guidance of

Prof. Rajarshi Roy
Dept. of Electronics and Elec. Comm. Engg.
IIT Kharagpur, Kharagpur

DEPARTMENT OF ELECTRONICS AND ELEC. COMM. ENGG.
INDIAN INSTITUTE OF TECHNOLOGY
KHARAGPUR – 721 302

Certificate

This is to certify that the thesis entitled "Distributed Deployment of Wireless Sensor Networks" being submitted by Mr. Jishnu Dey, Roll No. 12EC10026 to the Indian Institute of Technology, Kharagpur in the academic session 2015-2016 is a bona-fide record of the work done by him under my supervision and guidance. This thesis is submitted in the partial fulfillment of the requirement for the award of Bachelor of Technology in Electronics and Electrical Communication Engineering.

The results embodied in this thesis have not been submitted to any other university or institute for the award of any degree or diploma.

Date:

Prof. Rajarshi Roy
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Acknowledgements

I express my deep sense of gratitude to my guide Prof. Rajarshi Roy for his valuable guidance and inspiration throughout the course of this work. I am thankful to him for his kind help, assistance and support and his motivation toward independent thinking. It has been a great experience working under him in a cordial environment.

I would like to thank all the professors at IIT Kharagpur who have been helpful in the course of the project at some stage.

Lastly, I would like to thank my parents for their motivation and support through the course of this work.

Jishnu Dey
12EC10026
IIT Kharagpur

Contents

Abstract

In recent times, Wireless Sensor Networks have been a major topic of research, due to their wide domain of applications. Wireless Sensor Networks can be deployed in an area under surveillance for continuous monitoring. However, sensor nodes are heavily constrained, in terms of energy and computational power. This discourages the use of centralized algorithms for such networks, as the robustness of the system is compromised. This work focuses on techniques for distributed deployment of sensor nodes for achieving coverage, which is an important performance metric for sensor networks. Two different distributed approaches have been taken for two types of coverage; namely Area and Target coverage. For the problem of Area Coverage, a game theory based approach is used which treats the sensors as self-interested agents trying to maximize their local objective functions. For the problem of Target Coverage, a special graph structure is used to maintain connectivity and achieve coverage for a single target, and then it is extended to multiple targets.

List of Figures

Chapter 1

Introduction

Wireless Sensor Networks are a collection of sensors, that have the task of monitoring of some parameter in the environment, such as temperature, pressure, humidity etc. usually in a distributed manner. These sensors, apart from their sensing abilities have various other capabilities which help them perform this task. They have some computation power, though limited, which allows basic processing of the sensed data. They have locomotive abilities, which allows them to deploy themselves according to the objective they wish to achieve collectively. Another important characteristic of these sensor nodes is the ability to communicate, either within themselves, or with a base station, which allows them to exchange information regarding their position as well as the sensed data.

Sensor networks find application in a broad spectrum of areas including military, health-care, surveillance and industry. A few examples of application of sensor networks can be noted as follows:

- They can be deployed in disaster relief operations to monitor the degree of hazard. For example, measuring the mechanical stress of buildings after an earthquake.
- They can be used for precision agriculture by monitoring the amount of fertilizers and pesticides in the soil, and sending proper signals when there is need.
- They find application in health-care for long term surveillance of chronically ill patients or post-operative period through wearables.
- They help in reducing energy wastage by measurement and control of humidity, ventilation and air-conditioning.
- They can be used to monitor points of interest on a periodic basis.

1.1 Structure of Wireless Sensor Networks

A widely accepted model that describes Wireless Sensor networks is defined below.[1] Let there exist n nodes in the network. The set $N = 1, 2, 3, ..., n$ denotes the indexes of the nodes of the system. The set T denotes the set of time instants when snapshots of the system are taken.

- There exists a placement function P, such that $P : N \times T \rightarrow [0, l]^d$ where $[0, l]$ denotes the range of the domain towards a coordinate axis, and d denotes the dimension of the space in which the sensors are placed. In this work, the values of d are taken as 1 and 2.

- There exists a sensing range function S, such that $S : N \rightarrow s_i$ where s_i denotes the radius up to which a sensor can sense data with reasonable accuracy. In order to cover a target, the sensor must be at a distance less than the sensing radius from the target.

- There exists a communication range function R, such that $R : N \rightarrow r_i$ where r_i denotes the radius up to which a sensor can transmit data with reasonable accuracy. This distance is calculated based on the largest δ that satisfies the inequality defined by [2]
$$\frac{p_i}{\delta^\alpha} \geq \beta$$
Here, p_i denotes the transmitting power of the i^{th} sensor node. α is the transmission power gradient and is set to 2 for ideal situations and 4 for real situations. β refers to the transmission quality and is usually set to 1.

- Finally exists is the communication graph, G, defined as $G = (N, E(t))$ where at any $t \in T$, there exists an edge $[i, j]$ if and only if $d(i, j) \leq r_i$ and $d(j, i) \leq r_j$ where $d(i, j) = d(j, i)$ and is the euclidean distance between the two nodes. The graph G is undirected in usual practice, i.e., $[i, j] \in E(t) \Leftrightarrow [j, i] \in E(t)$.

There are some inherent weaknesses in this model. The assumption of a uniform radial coverage does not hold true for most real applications in the presence of obstacles. The sensing and communication radii are susceptible to external environment factors and interference from other nodes. However, incorporating these facts into the network model makes it difficult to perform simulations and generate general results. Thus, the above described model is widely used for analysis and simulation studies of wireless sensor networks.

1.2 Coverage in Wireless Sensor Networks

Coverage is Wireless sensor networks can simply be thought of as a metric, that measures how well the area under surveillance is being monitored. It determines the Quality of Service (QoS) of the deployed network, and tries to find weak points in the current configuration, i.e., the areas that are currently not being monitored properly. This can help in formulation of strategies for future deployment schemes.

There are mainly two notions of coverage which are discussed below:

- **Physical Coverage:** Under this notion, a point is said to be covered, if it lies within the sensing radius of any of the sensors deployed in the field. The minimum euclidean distance of the desired point from nearby sensors (or all) is calculated, and if it falls below the sensing radius of the corresponding sensors, it is said to be covered.

  A variant of this notion is k-coverage, where the k smallest distances are calculated, all of which should be less than the sensing radii of the corresponding sensors. This is important, as sensors are generally energy constrained, and are often prone to failure. Having k sensors monitoring each point brings robustness to the system.

- **Information Coverage:**[3] Under this notion, a point can be covered, even if it is not within the sensing radius of any of the sensors. As from the definition of sensing radius, it is the distance up to which a sensor can sense with reasonable accuracy. Beyond this radius, the proportion of noise in the data exceeds the acceptable threshold, making it difficult to recover the data. However, if multiple sensors sense the same data, with varying degrees of noise affecting the data, then the mean of the data from all these sensors can give a fairly accurate estimation of the actual parameter.

  This is explained by the Central Limit Theorem, which states that the arithmetic mean of N independent random variables, each having a finite mean and variance, converges into a normal distribution. Under added conditions of the random variables be identically distributed as a normal distribution with mean μ and variance σ^2, the resulting mean is a normal distribution with mean μ and variance σ^2/n. Thus, as the variance decreases, the uncertainty in the measured data reduces, allowing for accurate measurement of the parameters through collaboration of multiple sensors.

Physical coverage can be classified into three major schemes, namely Area, Target and Path Coverage.[4]

1.2.1 Area Coverage

This Coverage aims to maximize the total area that is monitored by the sensors in the field where they are deployed. It is inspired from the Circle Coverage Problem, where the aim to place a set of circles in a plane, with the aim to maximize the union of areas that fall under the circles. The problem can be formally defined as follows:[5]

$$\begin{aligned} &\mathit{maximize} && \left(\bigcup_{i=1}^{k} \bigcup_{j=n_i}^{n_{i+1}-1} C_{r_i}(x_j, y_j) \cap A \right) \\ &\mathit{subject\,to} && (x_i, y_i) \in A, \qquad i = 1, 2, ..., n \end{aligned} \tag{1.1}$$

The equation is explained as follows. The aim is to maximize the coverage of the domain A. There are k types of sensors, where each type has a sensing radius r_i and there are n_i number of each type. The coverage by the j^{th} sensor of the i^{th} type is given by the intersection of its sensing area with the domain. The total coverage is given by the union of all the individual coverages. An illustration for the problem is shown in Figure 1.1:

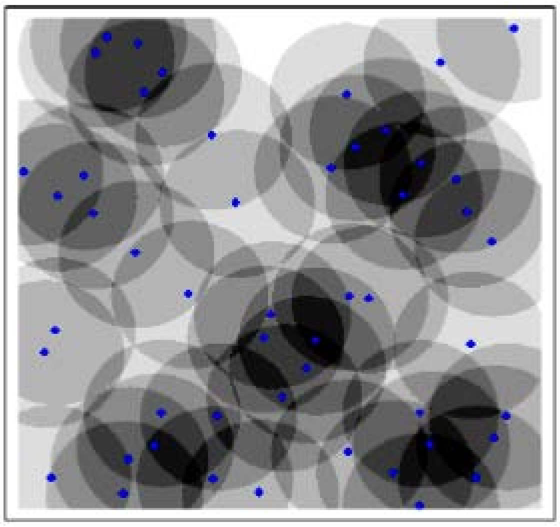

Figure 1.1: Modeling of the Area Coverage Problem

1.2.2 Target Coverage

Given a set of targets in the sensing field with known locations, the objective of this problem is to find a placement strategy such that all the targets are covered using the minimum number of sensors possible. It is inspired by the Art Gallery Problem, where the aim is to place minimum number of cameras to make the gallery thief-proof, by ensuring that all the displays are monitored by at least one camera.

There are two variants to this problem. In the first case, when the distances between the targets are comparable to the sensing radii, the problem becomes equivalent to that of the Maximum Set Cover (MSC), which can be defined as selecting k sets from a given number of sets with the aim of maximizing the total cardinality of the sets. It is a classical problem of computer science. An illustration of this formulation is depicted below in Figure 1.2:

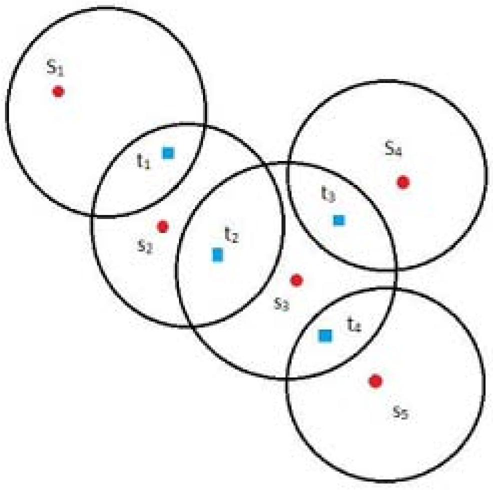

FIGURE 1.2: Modeling of the Target Coverage (MSC) Problem

In the other case, when the sensing radii are much smaller than the inter target distances, it becomes important to devise placement strategies which ensure connectivity between the sensors, so that the information can be propagated to the desired destination.

1.2.3 Barrier Coverage

The goal of this coverage problem to prevent undetected penetration through a barrier. It is inspired from the Robotic systems coverage, which has a similar objective. A variant of this is the sweep coverage, where the barrier is mobile in nature.

1.3 Distributed Deployment

There are two classes of algorithms, centralized and distributed, which are briefly explained:

- **Centralized**: These class of algorithms are dependent on a central node, which is much powerful compared to the other nodes. It has the responsibility of running the algorithm and processing the data that it receives from other nodes, and then transmit the same back to the nodes, allowing them to take respective actions. The advantages of this class of algorithms is the processing power of the central node, which allows for complex calculations to be done faster. Also, centralized algorithms are relatively simpler compared to their distributive counterparts. However, centralized algorithms suffer from certain drawbacks. A major one of them is the heavy reliance on the central node by the other nodes. If this node fails, the entire system comes to a standstill. Also, communication requirements are excessive for these type of algorithms, as all the data needs to be transmitted back and forth to the central node.

- **Distributed**: These class of algorithms do not require any central node. Each node can compute its part of the algorithm and take decisions based on it. For this purpose, it can communicate with sensors within its communication radius. These systems are more robust and more energy efficient compared to centralized systems. However, design of these algorithms are more challenging, with issues like constraints on computation power, synchronization etc.

Also there are two classes of sensors, mobile and immobile. Immobile ones are those which have no locomotive capability. So for placing these sensors, their optimal locations need to computed in advance, and the sensors be placed at those locations. These systems are not very effective in disaster intervention, or unmanned applications. Another type are the mobile sensors, which can move from one point to another. These sensors can either be dropped from the sky, or released from a base station, from where they can deploy themselves to achieve coverage based on either a centralized or a distributed algorithm.

1.4 Focus of this work

This work is aimed at the distributed deployment of mobile sensors, for achieving various types of coverage. Firstly, the problem of area coverage is considered, and using the approach of a state based potential game, a heuristic for distributed deployment is developed. Next, the problem of target coverage is considered, where the aim is to deploy sensors to several target locations while maintaining connectivity with a base station. This is achieved by considering a special graph structure over the sensor nodes.

Chapter 2

Area Coverage

A multi-agent system is defined as a set of independent agents having local objective functions which they seek to achieve. The class of optimization procedures that revolve around multi-agent systems aim to design these local functions such that the resulting global behavior is in accordance with the desired system objective. This process requires two steps, the first one being defining proper local objective functions which satisfy the above criteria, and second, designing a distributed algorithm which allows these agents to achieve these objective functions. Game theory is an effective tool in this process, as using a specific class of games, this process can be decoupled.[6]

The model of sensor networks, stated in the previous chapter, can be thought of a multi-agent system. In this process, the system level objective of coverage needs to be modified into local functions which can contribute towards a desired global behavior. The system level objective here, is the problem of area coverage, as defined in section 1.2. The aim is to maximize the coverage of the area in the domain given a set of sensors, with a fixed sensing and communication radii. For the maximization problem to be solvable through convex optimization algorithms, the function should satisfy convexity property.

2.1 Problem Formulation

For the ease of implementation, coverage in only one dimension is considered in this application. The objective function ϕ is defined below. The set $[x_1, x_2, ..., x_n]$ denotes the coordinates where the sensors are currently placed. the sensing radii is given by s_i. The domain is represented by an array A of size N, initialized to 0's. Coverage by a sensor is given as $1[x_i - s_i, x_i + s_i]$, where 1 is the indicator function, turning the corresponding array element to 1.

$$\phi(x_1, x_2, ..., x_n) = \frac{\sum_{i=1}^{N} A[i]}{N} \tag{2.1}$$

To determine the convexity of this function, a simulation was performed for just two sensors trying to cover a domain. The plot obtained from the simulation is shown below:

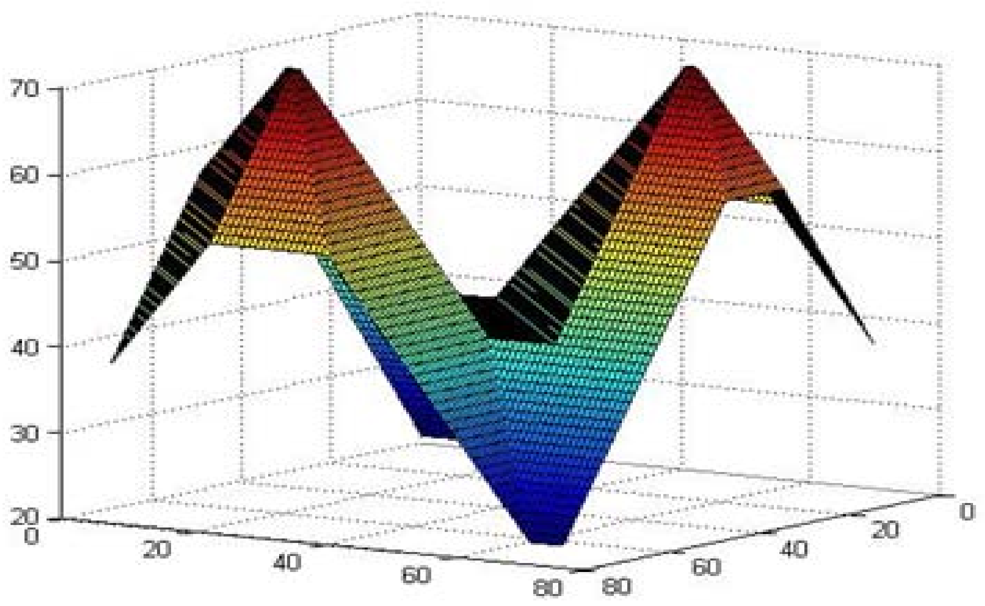

FIGURE 2.1: Plot of Coverage function for 2 variables

Consider the Figure 2.1. There appear to be multiple peaks, contradicting the notion of a concave function. But carefully looking, if we consider the a subset of the domain, either $[x_1 > x_2]$, or $[x_1 < x_2]$, we find that in these subsets, the function is strictly concave. Moreover, looking carefully, we find that each of those peaks is a permutation of one another, and has the same peak value. So, once the function is initialized at some random point, the current point on the domain shouldn't go out of the subset it is allotted. In other words, the order should be maintained, i.e., in the process of optimization, no point should jump over any other point. This ensures that the function eventually falls into the local maxima in its subset, thus implying the reachability of the global maxima.

In order to design the local objectives which satisfy the desired global behavior, we use a specific kind of game, called the State Based Potential Game, where the aim for each of the agent is to maximize or minimize its local objective function, which is designed in a manner to satisfy a global potential function.

2.2 State Based Potential Game

For a multi-agent system of N agents, a State Based Game is a particular class of games, where exists an underlying state space X. All agents $i \in N$ have a state dependent set of actions, denoted by $A_i(x)$, $\forall x \in X$. Also, the local objective functions are state dependent, given as $J_i : X \times A \to \mathbb{R}$, where $A = \prod_i A_i$ and $A_i = \prod_x A_i(x)$. The state changes are determined through a transition function given as $f : X \times A \to X$.

A subset of the class of State Based Games is the State Based Potential Games, where the local objective functions are defined as to satisfy a global potential function.[7] Mathematically, it can be described as follows. Assume the existence of a global potential function $\phi : X \times A \to \mathbb{R}$. The design of the game should be such that for every state action pair $[x; a] \in X \times A$:

- For any player $i \in N$ and action $a'_i \in A_i(x)$

$$J_i(x; a'_i; a_{-i}) - J_i(x; a) = \phi(x; a'_i; a_{-i}) - \phi(x; a) \tag{2.2}$$

- The potential function satisfies $\phi(x; a) = \phi(\tilde{x}; 0)$ where $\tilde{x} = f(x; a)$.

Assume a State Based Potential Game with global function ϕ. A single state equilibrium is reached for the state-action pair $[x^*; a^*]$ if $[x^*; a^*]$ satisfies $a^* = argmin_a \in A(x)\phi(x^*; a)$. If $[x^*; a^*]$ also satisfies $x^* = f(x^*; a^*)$, then $[x^*; a^*]$, then it is said to be a recurrent state equilibrium.

2.3 Design of the Game

There are various components of the State Based Potential Game which is applied to the problem of area coverage. This game was designed by Li and Marden.[8] Details are followed in the next subsections.

2.3.1 State Space

The state space X, is the set of all states x. Each state x is defined as a tuple $x = (v; e; G)$, where $v = (v_1, ..., v_n) \in R_n$ is a vector which determines the values that each node has taken. $e = (e_1, ..., e_n)$ is the matrix of estimation terms, where each $e_i = (e_i^1, ..., e_i^n) \in R_n$ is a vector of estimation of the value profile by the agent i. The term e_i^k is agent i's estimation of agent k's value v_k. G is the undirected communication graph, pertaining to the sensor network model discussed in the previous chapter. It is represented as a vector N_i for agent i.

In context of this application, the value profile is nothing but the coordinates occupied by the sensor nodes. The estimation matrix is the estimate of the position of the sensors by other sensors. These estimations may be correct, or may not be correct. But in order to reach equilibrium, the sensors must tend to increase the accuracy of their estimates.

2.3.2 Action Sets

The set of actions for a State Based Game is state dependent, denoted by $A_i(x)$. These actions allow the agents to change their values as well estimates of other nodes through communication with agents they are connected to, i.e., those with which they share an edge in G. The action of an agent i is defined by the tuple $a_i = (\hat{v}_i; \hat{e}_i)$ where $\hat{v}_i \in \mathbb{R}$ is the change in the agent's value v_i. $\hat{e}_i := \{\hat{e}_{i \to j}^k\}_{j \in N}^{k \in N}$ is the change in the agent's estimation terms e_i where $\hat{e}_{i \to j}^k \in \mathbb{R}$ is the term which represents the part of the estimation that player i passes to j regarding to the value of player k. It is to be noted that $\hat{e}_{i \to j}^k = 0, \forall j \notin N_i, k \in N$, as a node is only allowed to communicate only with the nodes in its neighborhood.

A change of value in this application implies that the agents move an equal distance, to a new position, with an aim to improve their local objective. The estimates change with the aim of making the estimations more accurate.

2.3.3 State Transition Rules

- The transition of the value profile v is captured by the local state transition rule of the form:

$$P_t^v(x, a) = P^v(x, a) = \{v_i + \hat{v}_i\}_{i \in N} \tag{2.3}$$

- The transition of the estimation profile e is captured by the local state transition rules of the form:

$$P_t^e(x, a) = P^e(x, a) = \{e_i^k + n\delta_i^k \hat{v}_i + \check{e}_i^k\}_{i,k \in N} \tag{2.4}$$

  where $\check{e}_i^k = \sum_{j \in N_i} \hat{e}_{j \to i}^k - \sum_{j \in N_i} \hat{e}_{i \to j}^k$ and δ_i^k is the indicator function.

- The state transition for G is mathematically defined as $P_t^G : X \times A \to \nabla(\bar{G})$ at each time t. Here $\bar{G}$ is the set of all undirected communication graph and $\nabla(\bar{G})$ is the set of probability distributions over this set. For this application, the graph is redrawn after every movement to incorporate the changes causes by the latest movement.

2.3.4 Invariance Property

Let $v(0) = (v_1(0), ..., v_n(0))$ be the initial values of the agents. The estimates $e(0)$ initially should satisfy $\sum_{i \in N} e_i^k(0) = n \cdot v_k(0)$ for each agent $k \in N$; thus, making the initial estimations dependent on the initialized values. This condition can trivially be satisfied as we can set $e_i^i(0) = n \cdot v_i(0)$ and $e_i^j(0) = 0$ for all agents $i, j \in N$ where $i \neq j$.

Looking at the state transition rules, we realize that a new estimation is created only when an agent moves $\hat{v}_i$, and the amount of estimation created is $n \cdot \hat{v}_i$, thus it's safe to say that the following condition is satisfied. It can be described as, $\forall t \geq 1$ and $j \in N$,:

$$\sum_{i=1}^{n} e_i^k(t) = n \cdot v_k(t) \tag{2.5}$$

2.3.5 Global Potential Function

Since for any State Based Potential Game, there exists a global function which gives a desired system behavior, designing such a global function is the key to choosing local objective functions compliant with the rules of the game. The function can be defined as $\Phi : X \times A \to \mathbb{R}$:

$$\Phi^{\phi}(x, a) = \Phi^{\phi}(x, a) + \alpha \cdot \Phi^{x}(x, a) \tag{2.6}$$

where

$$\Phi^{\phi}(x, a) = \sum_{j \in N} \phi(\tilde{e}_j^1, \tilde{e}_j^2, ..., \tilde{e}_j^n) \tag{2.7}$$

$$\Phi^{x}(x, a) = \sum_{j \in N} \sum_{k \in N} (\tilde{e}_j^k)^2 - n \cdot \sum_{j \in N} (\tilde{v}_j)^2 \tag{2.8}$$

$\tilde{v} = P^v(x; a)$ and $\tilde{e} = P^e(x; a)$.

This satisfies the property of Potential Games discussed in the previous section. Also, it's seen from the definition of the function that it is convex in nature, given that ϕ is also chosen to be convex. Since it is convex, as shown in the section 2.1 (actually $1 - \phi$ is convex, but that is a trivial fact), it is an appropriate function for an optimization problem.

2.3.6 Agent Cost Functions

Now that we have a global potential function, the local objectives can be picket out by taking the components from the global functions that are of concern to a a local agent. Following the lines of the global function, the local objective is of the form

$$J_i(x,a) = J_i^{\phi}(x,a) + \alpha \cdot J_i^{e}(x,a) \tag{2.9}$$

where $J_i^{\phi}(\cdot)$ represents the component related to the objective function, which in our case is already defined in section 2.1; $J_i^{e}(\cdot)$ represents the component based on the state x; and α is a which trades off the two components, giving proper weightage to both the components. The definition of each of the components is as follows: for any state $x \in X$ and action profile $a \in \prod_{i \in N} A_i(x)$:

$$J_i^{\phi}(x,a) = \sum_{j \in N_i} \phi(\tilde{e}_j^1, \tilde{e}_j^2, ..., \tilde{e}_j^n) \tag{2.10}$$

$$J_i^{e}(x,a) = \sum_{j \in N_i} \sum_{k \in N} (\tilde{e}_j^k)^2 - n(\tilde{v}_i)^2 \tag{2.11}$$

where $\tilde{v} = P^v(x;a)$ and $\tilde{e} = P^e(x;a)$.

2.3.7 Nash Equilibrium

A state action pair $[x;a] := [(v;e;G);(\hat{v};\hat{e})]$ for the above mentioned game is declared as a stationary Nash equilibrium in the game if and only if the following conditions hold:

- The value vector v is optimal;
- The estimation matrix e satisfies the condition $e_i^k = v_k$, $\forall i,k \in N$;
- The change in value vector satisfies the condition $\hat{v}_i = 0$, $\forall i \in N$;
- The change in estimation matrix satisfies the condition $\hat{e}_{i \to j}^k = 0$, $\forall i,j,k \in N$.

2.3.8 Application to Area Coverage

The problem is modeled as taking a large array of integer coordinates, and placing sensors randomly on some of the coordinates. Each sensor has a predefined sensing radius and a communication radius. The sensors can cover the area left and right to it within its sensing radius, and can exchange information with sensors within its communication radius. The initialization of the algorithm is done randomly, in accordance with the invariance property. Thus initially, every sensor has a nil estimate of the positions of the other sensors.

Since the ϕ defined in section 2.1 is the coverage we try to maximize, for the minimization problem we choose the objective as $1 - \phi$. On every iteration, the sensors try to correct their positions by trying to minimize their local objective functions, as given by a distributed algorithm. It is noteworthy that this minimization is based on their estimates of the location of other sensors, and not the actual locations of the other sensors. The sensors also exchange information about their estimates with other nodes within their communication radius, with an attempt to improve their estimates of the location of the other sensors.

2.4 Distributed Algorithm

Since the State Based Potential function $\phi(x; a)$ is convex over $a = (\hat{v}; \hat{e})$, each of the agents can use a gradient descent algorithm to reach the minima of their local objectives. When all the agents do this, it ensures that the system-wide minima is obtained, and thus, this leads to a distributed algorithm where each self interested agent behaves independently of others. The gradient descent algorithm for the agent i is given as follows:

1. Each agent i can be initialized randomly as $v_i(0)$ and further set $e_i^i = n \cdot v_i(0)$ and $e_i^k(0) = 0 \; \forall k \neq i$ in accordance to the invariance property. Set $t = 0$;
2. At each time $t \geq 0$ the action $a_i(t) = (\hat{v}_i(t); \hat{e}_i(t))$ selected by each agent i given the state $x(t) = (v(t); e(t); G(t))$ is:

$$\begin{aligned}
\hat{v}_i(t) &= \left[-\epsilon \cdot \frac{\partial J_i(x(t), a)}{\partial \hat{v}_i} \bigg|_{a=0} \right]^+ \\
&= \left[-\epsilon \cdot \frac{\partial J_i(x(t), a)}{\partial \tilde{e}_i} \times \frac{\partial \tilde{e}_i}{\partial \hat{v}_i} \bigg|_{a=0} \right]^+ \\
&= \left[-\epsilon (n \phi_{i|_{e_i(t)}} + 2n\alpha(e_i^i(t) - v_i(t))) \right]^+
\end{aligned}$$

$$\begin{aligned}\hat{e}^k_{i\to j}(t) &= -\epsilon \cdot \frac{\partial J_i(x(t),a)}{\partial \hat{e}^k_{i\to j}}\bigg|_{a=0} \\ &= -\epsilon \cdot \frac{\partial J_i(x(t),a)}{\partial \tilde{e}_i} \times \frac{\partial \tilde{e}_i}{\partial \hat{e}^k_{i\to j}}\bigg|_{a=0} \\ &= \epsilon(\phi_{k|_{e_i(t)}} - \phi_{k|_{e_j(t)}} + 2\alpha(e^k_i(t) - e^j_i(t)))\end{aligned}$$

where $[\cdot]^+$ represents the projection onto the closed convex set $A^{\hat{v}}_i(x)$. This projection in this formulation is the nearest integer in the permissible set, as the coordinates can only be integers. ϵ is the step-size of gradient descent. Since each agent i can select its own action using local information, the algorithm is truly distributed in nature. Note that $\phi_{k|_{e_i(t)}}$ is given by $\frac{\partial \phi(e_i)}{\partial e^k_i}$.

3. The state transitions to $(v_i(t+1); e_i(t+1))$ are done in accordance to the State transition rules discussed earlier. These depend only on local information. The communication graph $G(t)$ is realized according to P^G_t.

4. Increase t by 1 and return to step 2.

As the algorithm proceeds, the estimation of the sensors, regarding the location of other nodes improves and tends to be more accurate. The sensors tend to adjust their locations as to reach a local minima (which as a result of the potential function to be the global minima). The algorithm is said to have fully converged, when the percent coverage reached a threshold as specified by the user. Thus by the game theoretic approach, at this, point, all the agents have the proper estimates of all other nodes, as well has they have minimized their local objective functions.

2.5 Simulation and Results

Various simulations were run with varying parameters for the 1D coverage problem. A few sample results have been shown as follows. The plots of coverage, position dynamics, and the estimation of a particular sensor made by all the sensors over a period of time. Also, the final location arrays and estimation matrices are listed.

Specifics of the simulation:

- Number of sensors: 5
- Sensing Radii: [25-30]
- Communication Radii: [40-75]

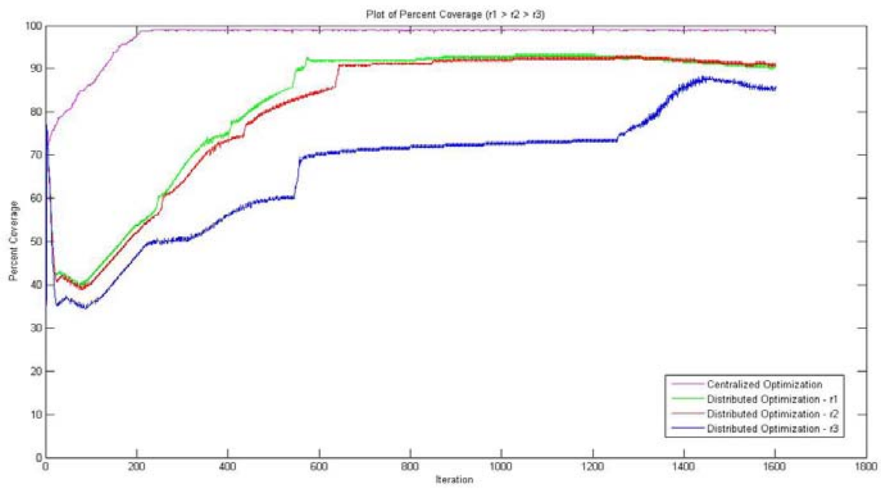

FIGURE 2.2: Plot of Coverage with iterations for various communication radii

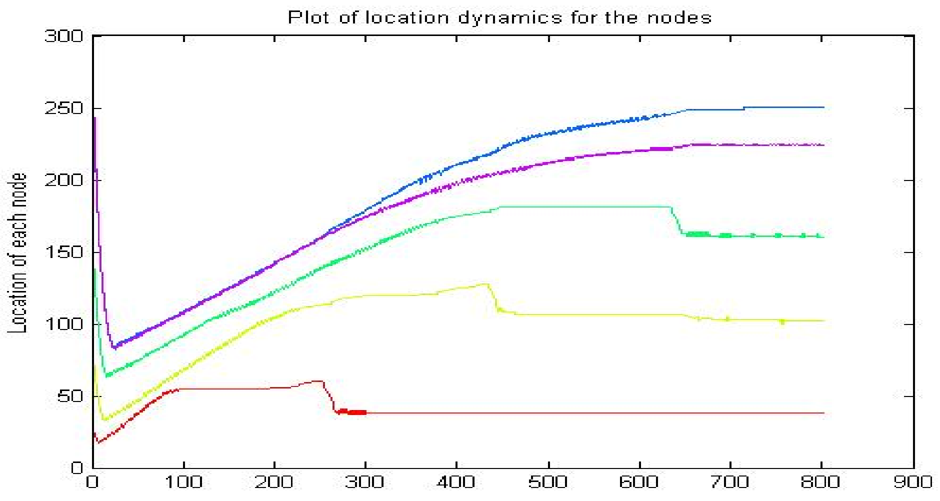

FIGURE 2.3: Plot of Location dynamics with iterations

Final Location Array:

$$\begin{vmatrix} 248 & 152 & 93 & 204 & 31 \end{vmatrix}$$

Final Estimation Matrix:

$$\begin{vmatrix} 248.0981 & 147.7525 & 90.9995 & 203.2841 & 32.5532 \\ 248.0291 & 165.7453 & 88.7874 & 202.4950 & 31.6493 \\ 247.9728 & 150.1231 & 96.4998 & 205.3793 & 27.6439 \\ 247.9233 & 147.5169 & 88.9461 & 204.5720 & 31.1855 \\ 247.9766 & 148.8622 & 89.7672 & 204.2697 & 31.9680 \end{vmatrix}$$

As it can be seen from Figure 2.2, coverage is almost achieved, with some margin of

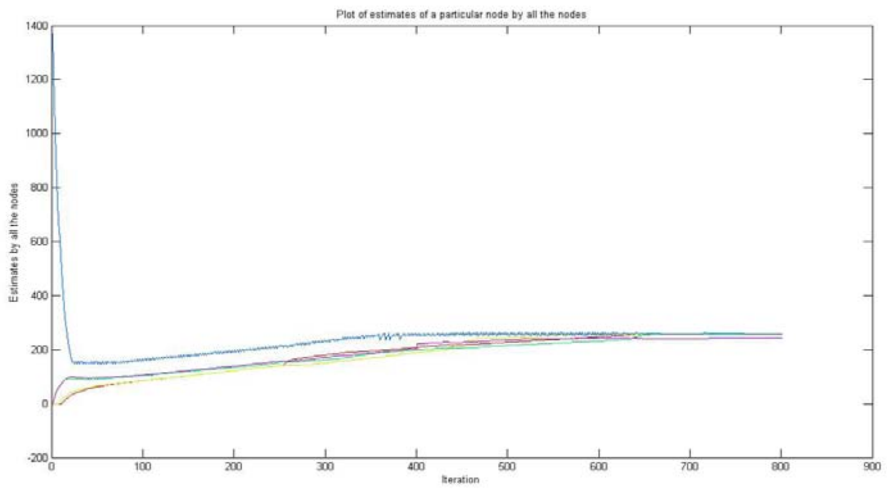

Figure 2.4: Plot of Estimates by the sensors for a sensor

error. The plot focuses on the trend when the communication radii is changed. As it increases, each of the sensors can interact with more agents, exchange information and reach convergence faster. For lower radii values, this process takes more time, and can often fall into loopholes where it gets isolated and can't achieve convergence at all.

Figure 2.3 shows the location dynamics, i.e., how the sensors change their position with time in order to achieve convergence. As seen, initially the sensors move very quickly due to rapid changes in values as their estimations are very wrong about other sensors. As this corrects with time, the sensors move on towards correct positions that will help them achieve maximum coverage.

Figure 2.4 shows how the estimates of all the sensors about a particular node change with time. The estimations start in accordance to the invariance property, and as the estimations get accurate, they converge on to the true value of the corresponding sensor.

For a particular simulation, the value vector v and the estimation matrix e are also listed. It can be seen that within limits of some error, the first two conditions of Nash Equilibrium is satisfied, which suggests that convergence has been achieved within an allowable limit of error.

Chapter 3

PoI Coverage with connectivity

Moving from the problem of Area Coverage, another issue is PoI(Point of Interest), or Target coverage, as discussed under various types of coverage. PoIs are of importance, because in certain scenarios, sensing the entire field is not important. The formalization of the problem is defined as follows:

Given a domain A, there are various PoIs scattered throughout the domain. There exists a base station, which acts as the starting point for all the sensors. Once the sensors are released, they must deploy themselves in a distributed manner, and ensure two criteria in doing so:

- All the PoIs should be covered by at least one sensor each.
- The sensors should maintain connectivity amongst themselves such that all the PoIs remain connected to the base station.

Firstly we consider the case of one PoI, and then move on to multiple PoIs. The aim here is to design a deployment strategy as to cover the PoI while ensuring that the sensors are connected with each other.

From the structure of sensor networks, as discussed earlier, we know that there exists an underlying graph structure for the nodes. Connectivity in sensor networks is ensured if the underlying graph is connected. There are a lot of connected graphs in literature, such as the Minimum Spanning Tree(MST), Relative Neighborhood Graph(RNG), Gabriel Graph(GG), Complete Graph(CG) etc. Moreover, any tree is a connected graph. The choice of the graph should be such that it minimizes the number of sensors required to cover the given PoI. For this purpose, we consider the concept of length stretch factor.

3.1 Length Stretch Factor

Let $G = (V, E)$ be a graph defined on an n-vertex set V with and edge set E. The distance in G between two vertexes $u, v \in V$ is total length of the shortest path between u and v and is denoted by $d_G(u, v)$. A subgraph $H = (V, E')$, where $E' \subseteq E$n is a *t-spanner* of G if for every $u, v \in V$, $d_H(u, v) \leq t \times d_G(u, v)$. The value of t is called the *Length Stretch Factor.*[9]

Ideally for this application, a graph is desired whose Length Stretch Factor is as large as possible. During the course of the algorithm, the graph will be redirected and stretched while maintaining connectivity. This will cause the graph to continuously evolve and spread out. Eventually, the goal is to transform the graph into a state which contains a path with $O(n)$ hops, which ensures maximum stretching out. It was shown in [10], that for a Relative Neighborhood Graph, the Length Stretch Factor is of the order $\Omega(n)$, while for a β-skeleton for $\beta \in [0, 2]$ (RNG is a 2-skeleton), the maximum Length Stretch Factor can be $n - 1$. For various other graphs, such as Gabriel Graph, the Length Stretch Factor is of the order of $\Omega(\sqrt{n})$, which is obviously suboptimal. Thus, This makes RNG a suitable choice for this application.

3.2 Relative Neighborhood Graph

The Relative Neighborhood Graph (RNG) was first discussed by Godfried T. Toussaint [11]. RNG is a connected graph. The connections in RNG leads to a graph which extracts a "perceptually meaningful structure" from the set of points. The RNG can be formally defined as:

Consider a set V of n distinct points in a plane: $V = 1, 2, 3, ..., n$. Two points u and v can be defined as relative neighbors, if $d(u, v) \leq max[d(u, w), d(w, v)]$ $\forall w = 1, 2, ..., n$, $w \neq u, v$. The Relative Neighborhood graph is constructed by forming edges between u and v if they are relative neighbors $\forall u, v = 1, 2, ..., n$ and $u \neq v$.

Figure 3.1 depicts the set of points in space where a node w can not exist, if nodes u and v are relative neighbors. Let $\delta = d(u, v)$ denote the Euclidean distance between the nodes. Firstly, $\delta \leq r_u$ and $\delta \leq r_v$ for an edge to exist between these two nodes, where r_u and r_v are the communication radii of these nodes. Now, consider circles of radius δ being drawn around each of the nodes u and v. The interior of each circle denotes the locus of points whose distance is smaller than δ. By the definition of RNG, any point w that lies inside the intersection of these interiors, depicted by the shaded region satisfies the criteria $d(u, v) \leq max[d(u, w), d(w, v)]$, and thus wipes out the edge

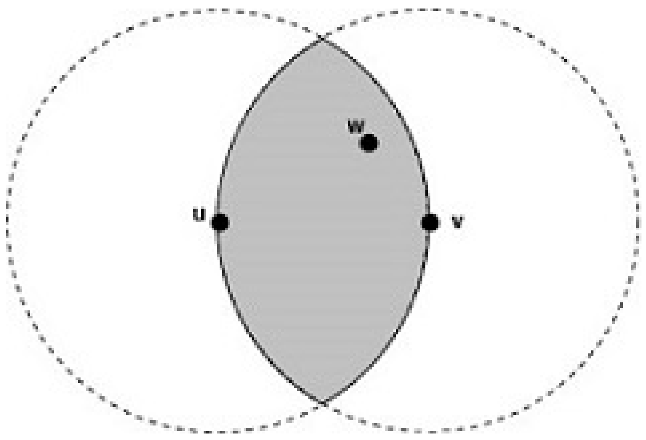

FIGURE 3.1: Relative Neighbor Illustration

that existed between u and v. Thus, u and v are relative neighbors, then there can not exist any point w inside the shaded region.

A basic algorithm for calculating RNG is described:

Algorithm 1 Basic RNG computation

```
$edge \leftarrow 0[n \times n]$
**for all** $u, v = 1, 2, ..., n$ **do**
  $d(u, v) \leftarrow \|(p(u) - p(v)\|_2$ {$p(u)$ denotes the position of the sensor}
**end for**
**for all** $u, v = 1, 2, ..., n$ **do**
  **for all** $w = 1, 2, ..., n$ **do**
    $d^w_{max} \leftarrow max[d(u, w), d(w, v)]$
  **end for**
**end for**
**for all** $u, v = 1, 2, ..., n$ **do**
  **for all** $w = 1, 2, ..., n$ **do**
    Search for $d^w_{max} \leq d(u, v)$
  **end for**
  **if** No such $w$ exists **then**
    $edge(u, v) \leftarrow 1$
  **end if**
**end for**
**return** $edge$
```

This algorithm is $O(n^3)$. This makes it a bad choice for implementation, even though its straight forward and can be easily implemented using local information as each node requires only one one-hop distances for computing its edges to neighbor sensors. Another algorithm was proposed by Toussaint, which uses the concept of Delaunay triangulation, which is discussed below.

3.2.1 Delaunay Triangulation

Given a set of points V, a Delaunay Triangulation of the nodes is a construction of a set of triangles over the given set, such that the circumcircles of none of the triangles contains another node in its interior. Delaunay triangulations tend to avoid small angle triangles, as these have higher chances of including other nodes in their interior.

An interesting aspect of Delaunay Triangulation is its relation to the Voronoi diagram, which is a partition of the given domain into a finite number of sets, such that the set enclosing a given vertex v is the set of all points in the domain which are the nearest neighbors of v. Delaunay Triangulation is simply the dual of the Voronoi diagram, and can be constructed by joining the pair of vertexes which are bisected by the lines of the Voronoi Diagram. An Illustration in Figure 3.2 helps to visualize:

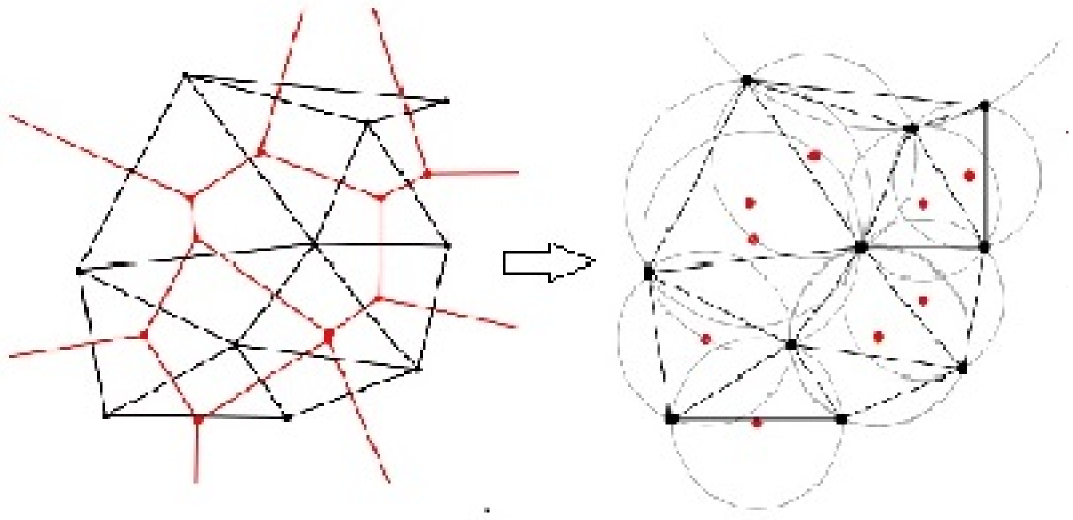

FIGURE 3.2: Relation between Voronoi and Delaunay

There are various algorithms for computing Delaunay Triangulation:

- **Flipping**: The Delaunay condition says that if two triangles share a common edge, then the sum of opposite angles should be greater than 180°. If the condition is not met, then flipping the common edge ensures that the condition is met. Number of edges is $O(n^2)$, and thus an algorithm with $O(n^2)$ can be constructed with creating any triangulation and flipping all edges till all triangles are Delaunay.

- **Incremental**: In this technique, one vertex v is added to the triangulation in each iteration. The triangles which contain the vertex v are searched for, which are divided into three parts, and then flip algorithm is applied to it. Considering the time complexity, this algorithm also runs in $O(n^2)$ time.

- **Divide and Conquer**: This algorithm first divides the given points into a number of small sets. The Delaunay Triangulation of these points are naïvely computed. These small triangulations can then be merged together in $O(n)$ time, giving an overall complexity of $O(nlogn)$.
- There are some other algorithms like the Sweep-line Algorithm which runs in $O(nlogn)$ time.

A theorem given by Toussaint concerning the relation between Relative Neighborhood Graph and Delaunay Triangulation is stated below:

Theorem 3.1. *The Relative Neighborhood Graph(RNG) is a subset of the Delaunay Triangulation(DT).*

Proof. Consider Figure 3.3, a and b are the points having an RNG edge. Since they are relative neighbors, the region R can not contain any other point. For RNG to be a subset of DT, a and b must have an edge in the DT too. For this, the length of the bisector of a and b must have a non zero length. Consider the points c and d. The intersection of the bisectors of $c - a$ and $a - d$ with that of $a - b$ reduces the length of bisector of $a - b$. But, this length can never be zero, no matter how close c and d are to a, because they lie on a circle and the angle will be non-zero, however small though. Thus the existence of a bisector in the Voronoi diagram confirms the presence of an edge in the DT. Thus, RNG is a subset of DT. □

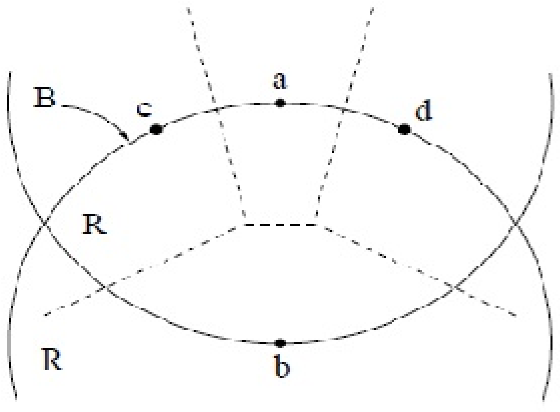

FIGURE 3.3: Illustration for proof of Theorem 3.1

3.2.2 Faster Algorithm for RNG Computation

Toussaint gave another algorithm for computation of RNG, based on this theorem, which runs in $O(n^2)$ time. The algorithm restricts the search space for edges of RNG to that of Delaunay only.

Algorithm 2 Fast RNG computation

```
Compute the Delaunay Triangulation for the set $V$ of nodes.
$edge \leftarrow 0[n \times n]$
for all $u, v = 1, 2, ..., n$ do
  $d(u,v) \leftarrow \|(p(u) - p(v)\|_2$ {$p(u)$ denotes the position of the sensor}
end for
for all $u, v$ such that (u,v) is an edge of DT do
  for all $w = 1, 2, ..., n$ do
    $d^w_{max} \leftarrow max[d(u,w), d(w,v)]$
  end for
end for
for all $u, v$ such that (u,v) is an edge of DT do
  for all $w = 1, 2, ..., n$ do
    Search for $d^w_{max} \leq d(u,v)$
  end for
  if No such $w$ exists then
    $edge(u,v) \leftarrow 1$
  end if
end for
return $edge$
```

This algorithm uses the fact that RNG is a subset of Delaunay Triangulation, and thus any edge of RNG is also an edge of DT. So, the search for edges of RNG is restricted to the edges of DT. The Delaunay Triangulation can be efficiently calculated in $O(nlogn)$ time, as discussed in the previous section. The Delaunay Triangulation results in $O(n)$ edges, which reduces the search space from $O(n^2)$ pair of points to $O(n)$. As a result, the overall complexity of this algorithm is $O(n)$.

3.3 Urquhart Graph

Consider three nodes from the set V, namely u, v and w. Let us claim that if an RNG is formed on V, there can not exist a triangle between the nodes u, v and w. To see this, without loss of generality, assume edge lengths are ordered as $(u,v) \leq (v,w) \leq (u,w)$. This contradicts the Relative Neighbor condition, that $(u,w) \leq max[(u,v),(v,w)]$. Thus, an edge can not exist between u and w.. Thus, the triangle can not be formed. Incorporating this idea into the Delaunay Triangulation leads to a graph structure that is called the Urquhart Graph[12], proposed as an $O(nlogn)$ algorithm for computing the Relative Neighborhood Graph.

The basic idea of the Urquhart graph is that RNG is a subset of Delaunay Triangulation, and since no triangles can exist in a RNG, removing the longest edge from each of the triangles in Delaunay Triangulation ensures the relative neighbor condition for all the points in V. However, in [13], Toussaint provided a counter example, which showed that the Urquhart algorithm restricts the search space for a given edge to only its two adjacent triangle, which is not sufficient to ensure the Relative Neighbor condition. Thus, this can lead to additional edges in the Urquhart Graph which ensures that $Urquhart\,Graph \subseteq Relative\,Neighborhood\,Graph$. But owing to the rare probability of the occurrence of such a scenario, especially in the case of randomized sampling, Urquhart Graph provides an excellent approximation of the Relative Neighborhood Graph. Thus, for computational purposes, Urquhart graph can be used in place of RNG, because of its smaller time complexity, $O(nlogn)$ as well as the simplicity of implementation. The algorithm for computation of Urquhart Graph is given below:

Algorithm 3 Urquhart Graph computation

```
Compute the Delaunay Triangulation for the set $V$ of nodes.
$edge \leftarrow 0[n \times n]$
**for all** Triangles $u, v, w$ in the DT **do**
  $edge(u, v) \leftarrow 1$
  $edge(v, w) \leftarrow 1$
  $edge(u, w) \leftarrow 1$
**end for**
**for all** Triangles $u, v, w$ in the DT **do**
  $(p1, p2) \leftarrow max[d(u, v), d(v, w), d(u, w)]$
  $edge(p1, p2) \leftarrow 0$ {$p1$ and $p2$ store the vertexes corresponding to longest edge}
**end for**
**return** $edge$
```

Now we discuss the time complexity of the algorithm. The computation of Delaunay Triangulation requires $O(nlogn)$ time. It leads to the creation of $O(n)$ triangles. Each triangle requires $O(1)$ processing time. So overall, the complexity is dominated by DT computation, and is $O(nlogn)$. This is significantly faster compared to the second algorithm for RNG computation, which is $O(n^2)$. Thus, with the margin of some error, the Urquhart Graph is a simple and fast implementation of the RNG. The margin of error using Monte-Carlo simulations is done next.

3.3.1 Margin of Error

Below is an illustration of the Relative Neighborhood Graph and the Urquhart Graph generated for a random set of 100 points. For the Urquhart Graph, the additional edges are represented by red lines, in addition to the blue lines in the original RNG. There are a total of 245 edges in the RNG as shown in Figure 3.4, while the number of edges in the UG is only 4 more, i.e., 249, as shown in Figure 3.5. This corresponds to an error close to 1.6%.

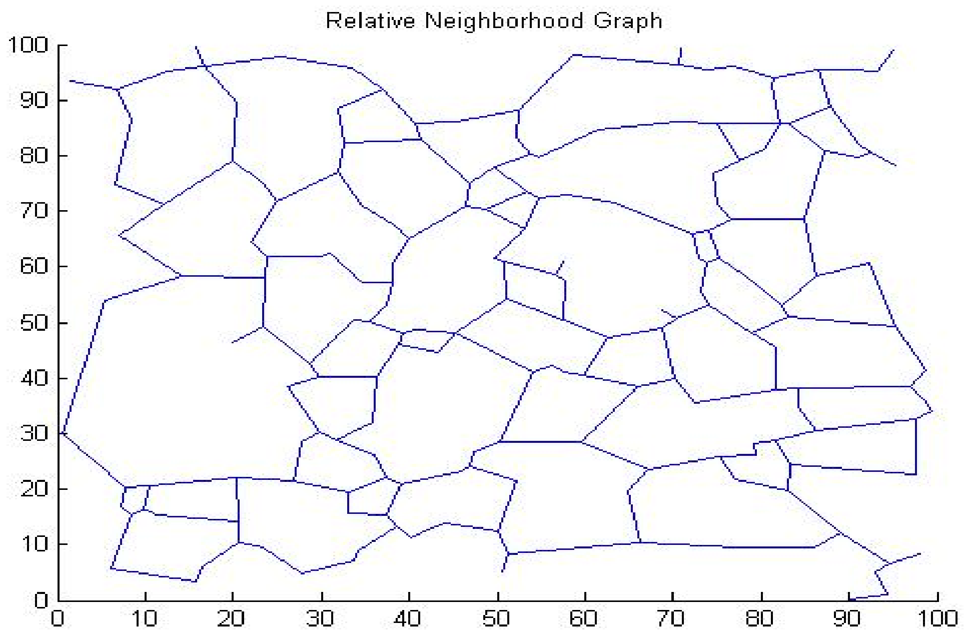

FIGURE 3.4: Illustration of Relative Neighborhood Graph

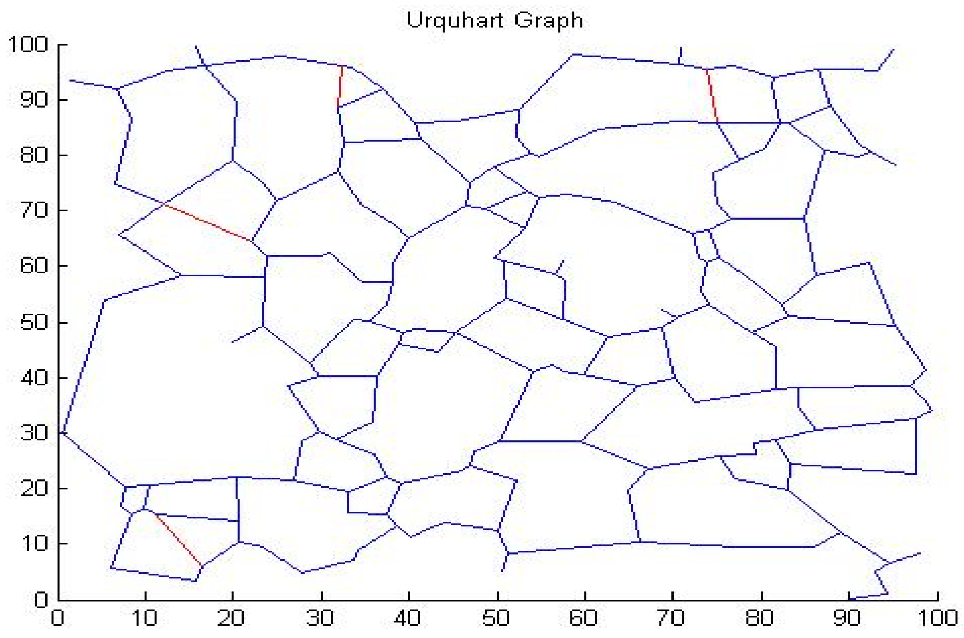

FIGURE 3.5: Illustration of Urquhart Graph

Monte Carlo Simulation were performed, ranging from few vertexes to thousands, and RNG and UG were generated for both. The difference in the number of edges were computed and the results plotted in logarithmic scale, mainly to highlight the lower error that is associated with small number of vertexes. The results are illustrated in Figure 3.6.

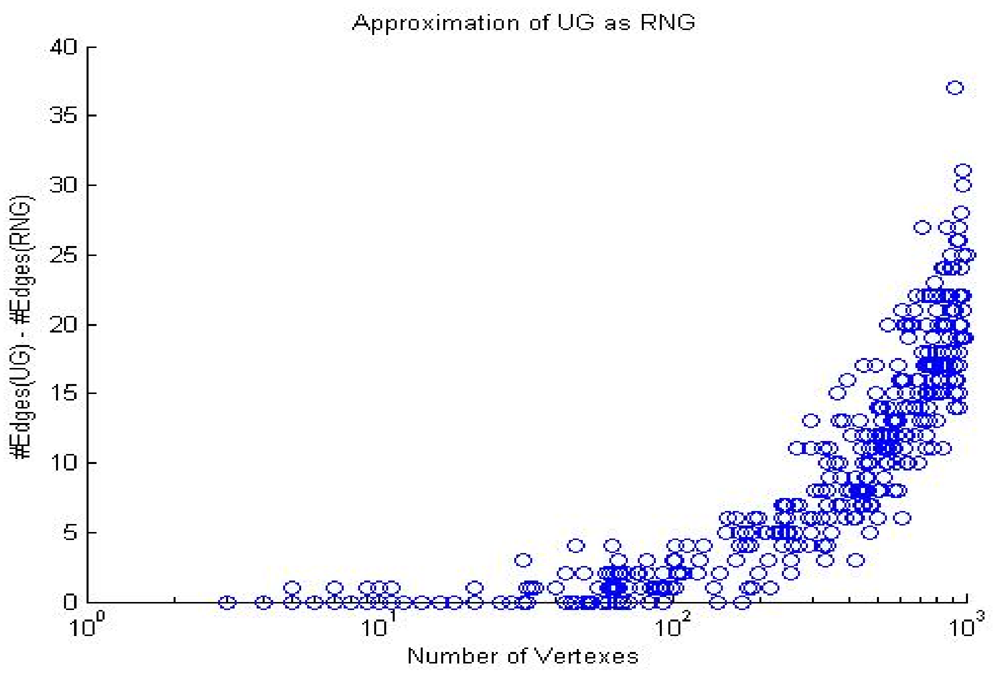

FIGURE 3.6: Illustration of RNG approximation by UG

As seen by the results, the number of extra edges in Urquhart Graph is only a small fraction of the total number of edges in the Relative Neighborhood Graph, and is approximately 2% for large number of vertexes. An interesting thing to note from the graph in Figure 3.6 is that for the number of vertexes less than 30, the difference between RNG and UG is rare, and they are exactly the same most of the times. This is important in the next subsection, where the Urquhart Graph will be calculated locally, and given that only a small number of nodes will surround any given sensor, the local computation of the Urquhart Graph will often be exactly the same as RNG.

3.3.2 Local Computation of UG

There are two main reasons which vouch for a technique for local computation of the Urquhart Graph:

- While the sensors are moving in the field with an aim to cover the desired target, the position of the nodes will change. This inevitably changes the inter nodal distances, and thus, the relative neighbor condition for many of the sensors. These changes must be incorporated into the existing UG, to create an updated one.

- The centralized algorithm as discussed will not work once the sensors have been deployed in the field, as they lose direct contact with the base station. Relying on the base station indirectly through message passing is a costly affair and should not be followed. A better alternative is to have an algorithm which allows for the UG to be computed locally, in a distributed manner.

The local computation of the Urquhart Graph must be carried out in such way that each node can calculate the edges coming out of it, using only the information available to it locally. The category of nodes which can contribute to this local information are those which lie inside the communication radius.

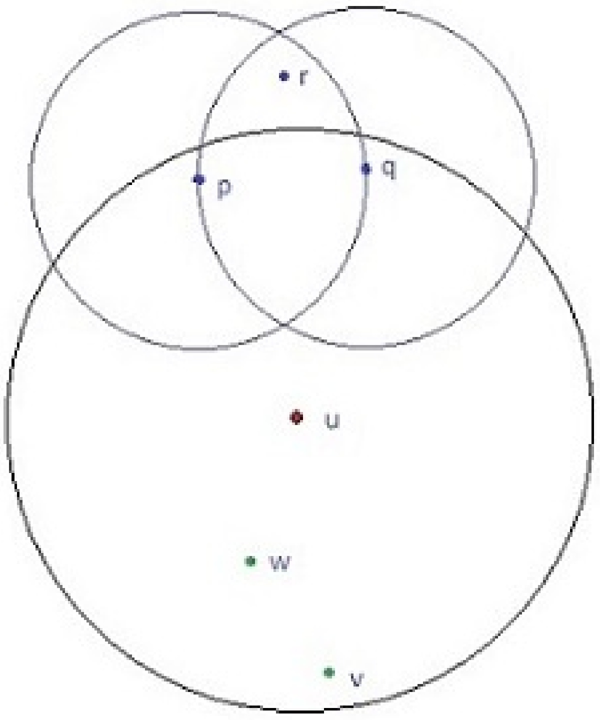

FIGURE 3.7: Neighborhood of a local sensor

There are a few points to consider here in Figure 3.7. Let us think from the perspective of the Relative Neighborhood Graph as the Urquhart Graph can be considered nothing more than a fast computation of the former. Consider the sensor in focus, u, which is locally computing its edges in the RNG. One thing that is given is that any vertex connected to u lies inside its communication radius. Also, since any edge that emanates out of u satisfies the relative neighborhood condition, whether an edge should be formed with a point v that lies inside the communication radius of u, can be confirmed by checking the Relative Neighbor condition only on all points w that lie inside the communication radius of u. Thus, all edges that can be formed by u which satisfy the relative neighbor condition can be computed only using the local information of u.

Now consider the points p and q that lie inside u's communication radius, and r, which does not. r violates the relative neighbor condition of p and q, thus not allowing an edge to form between them in the global RNG. However, when u computes the local RNG for the set of points in its communication range, it will form an edge between them. This is contradictory, which proves that when a local sensor computes the RNG, the local graph has no accountability for any edges other than those which emanate from the local sensor itself. So, the basic idea of the distributed algorithm is, to compute the local RNG for the given sensor, and then keep only those edges which are connected to the local sensor.

Algorithm 4 Local Urquhart Graph for node u

```
$V_u \leftarrow \{v \in V \mid d(u,v) \leq r_u\}$ {$r_u$ is $u$'s communication radius}
$k \leftarrow |V_u|$
$edge \leftarrow 0[k \times k]$
Compute the Delaunay Triangulation for the set $V_u$ of nodes.
for all Triangles $a, b, c$ in the DT do
    $edge(a,b) \leftarrow 1$
    $edge(b,c) \leftarrow 1$
    $edge(a,c) \leftarrow 1$
end for
for all Triangles $a, b, c$ in the DT do
    $(p1, p2) \leftarrow max[d(a,b), d(b,c), d(a,c)]$
    $edge(p1,p2) \leftarrow 0$ {$p1$ and $p2$ store the vertexes corresponding to longest edge}
end for
$edge_u \leftarrow edge[u, 1...k]$
return $edge_u$
```

This algorithm runs is $O(klogk)$ time, where k is the number of nodes inside the communication radius of u.

3.4 Deployment Algorithm

The deployment process starts with all the sensors inside the base station. It has a centralized node, and initially all the sensors are within its communication radius and can communicate with it directly. This allows the base station to do some preprocessing. Initially, all the nodes are within the communication range of the base station, which indirectly implies that they are also within the communication range of one another. Thus the base station lays out the RNG (or UG) for the entire set, connecting all the nodes. Also, the base station has the knowledge of the location of the Point of Interest, which it imparts to all the sensors.

The number of sensors to be released can easily be precomputed based on the distance of the PoI. The number of sensors should be enough to maintain the link from the base station to the PoI. Once the preprocessing is done, sensors are deployed out into the field, and they lose contact with the base station, and henceforth, need to make computations and take decisions on their own. So, the deployment should be distributed in nature. Also, it needs to be asynchronous, for the lack of a global clock for the system.

Once the sensors are deployed, each sensor carries out a set of tasks in making its move. First, it calculates the local Urquhart Graph, to update the edges that emanate from it. Next, among the nodes directly connected to it, searches for the one farthest from it. Then, it takes a decision for movement, while ensuring that the link between it and the farthest node connected to it is not broken. If this condition is ensured, the direction can be anything. For making the process fast though, it is most beneficial if the nodes move directly towards the PoI.[14] The maximum distance that the nodes are allowed to move can be seen from the Figure 3.8:

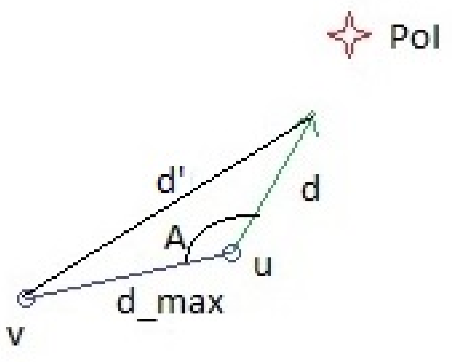

FIGURE 3.8: Movement Vector for node u

As seen from the figure, let us say that u is the node in focus, which has to take its movement decision. v is the node farthest to it, its distance given by d_max. r_u is node u's communication radius, and if d' is the new distance between the nodes u and v, then $d' \leq r_u$ is a necessary condition. Using cosine rule, we can write:

$$d' = \sqrt{d^2 + d_{max}^2 - 2 \cdot d \cdot d_{max} \cdot \cos A}$$

Using $d' \leq r_u$, and the fact that d' attains its maximum for $A = 180°$, we get:

$$\sqrt{d^2 + d_{max}^2 + 2 \cdot d \cdot d_{max}} \leq r_u$$
$$d + d_{max} \leq r_u$$
$$d \leq r_u - d_{max}$$

Thus, the maximum distance that a node can travel is $r_u - d_{max}$ where d_{max} is the distance of the farthest node that u has an edge with. It should be noted that this is not a tight bound, but a general bound keeping the displacement independent of direction of the movement, i.e., irrespective of any direction the sensor moves, if it doesn't travel a distance more than this, it will remain connected to every node it was previously attached to. It can be noted that it is most beneficial for if the sensors move directly towards the PoI. This allows them to cover distance from the PoI as fast as possible, and stretches the graph towards the PoI. With all this, the algorithm for the deployment process can be laid out now. The PoI is covered when at least one of the sensors has the PoI within its sensing range.

Algorithm 5 Deployment process for single PoI

```
$n \leftarrow \lceil \| \overrightarrow{t} \| / r \rceil$ {r is the commom communication radius}
Base station computes the Urquhart Graph using Algorithm 3
PoI is located at $\overrightarrow{t}$
while PoI is not covered do
    Select a sensor $u$ at random {$u$ is located at $\overrightarrow{u}$}
    $u$ updates its local edges of the Urquhart Graph using Algorithm 4
    $V_u \leftarrow \{v \in V \mid d(u,v) \leq r_u\}$
    $k \leftarrow |V_u|$
    $d_{max} \leftarrow 0$
    for all v=1,2,...,k do
        $d_{max} \leftarrow max[d_{max}, d(u,v)]$
    end for
    $disp \leftarrow (r_u - d_{max} - \epsilon)$ {$\epsilon$ is a small number}
    $\overrightarrow{dir} \leftarrow (\overrightarrow{t} - \overrightarrow{u})$
    $\overrightarrow{mov} \leftarrow disp \times \overrightarrow{dir} / \| \overrightarrow{dir} \|$
end while
```

3.4.1 Simulation

Figure 3.9 shows the result of the simulation of the above algorithm. The algorithm allots 14 sensors for a PoI located at $(16, 18)$. The communication radii of the sensors is taken as 2. Sensing radii of the sensors is set at 1. The algorithm is initialized by placing all the sensors within the communication radius of the base station, which acts as a central node and defines the initial graph structure over the nodes. Once that is done, the sensors move out and deploy themselves according to Algorithm 5.

There are two practical issues of implementation. First is to avoid infinitesimal movements by the sensors, as the coordinates are in $\mathbb{R}$. To avoid this, a threshold of 0.01 is chosen. If the displacement is below this value, the sensor doesn't move. The next issue is of termination of the algorithm. The algorithm terminates when the PoI is within the sensing range of at least one of the sensors.

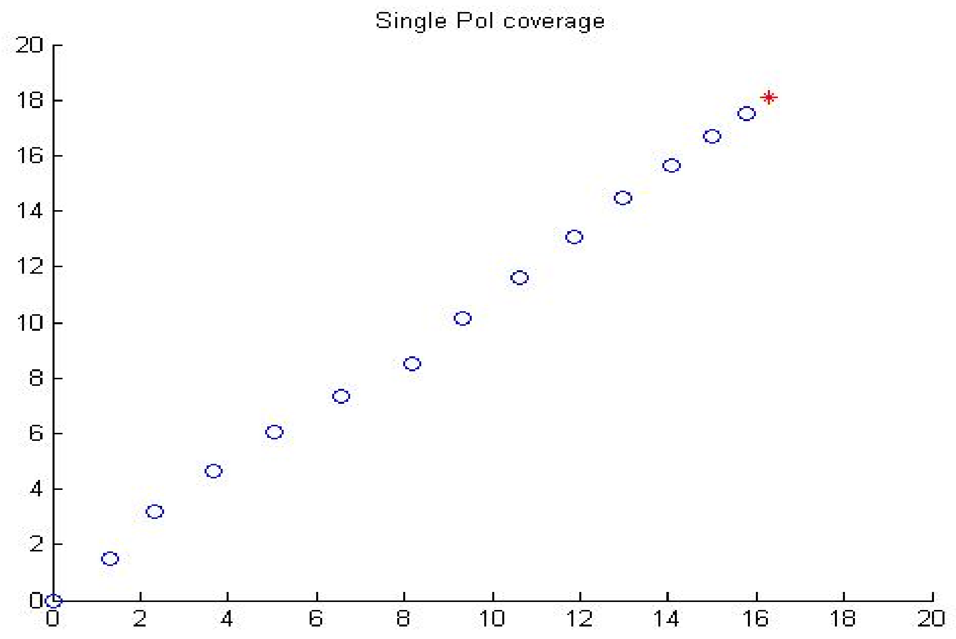

FIGURE 3.9: Simulation for single PoI coverage

As seen from the figure, the algorithm achieves coverage of the PoI. The sensors have stretched out with a stretch factor of $n - 1$, showing the utility of the RNG structure.

Chapter 4

Multiple PoIs

Now that a deployment procedure has been laid out for covering a given Point of Interest while maintaining connectivity, the next issue is covering multiple such PoIs. A simple scheme that can be used for this deployment is just to allot a set of sensors to each PoI, and treat all PoIs as independent of each other. Though a very simple approach, this method will require a huge number of sensors, and that grows proportionally with the distance of the PoI from the base station. So, such a naïve method should not be used.

An approach that attempts to reduce the number of sensors is club them, i.e., propagate a common chain of sensors up to a distance, and then spread them out to cover each of the individual PoIs. An important criterion to consider here is that sensor nodes are highly energy constrained, running on limited battery power, and can not cover a large distance to achieve the objective. An important objective of the mechanism design is to reduce the maximum distance that any sensor has to travel in order to cover all the PoIs. Formally this can be defined as:

Let $\rho = max(\frac{\sum disp(u_i)}{\|p(t)\|_2})\ \forall i \in sensors$, where u_i is a sensor i covering any target t and $p(x)$ denotes the position of x in $2D$ plane. ρ this denotes the ratio of maximum distance traveled to displacement for covering any target by a sensor. The aim of the design is to reduce ρ as much as possible, as this ensures that a sensor reduces the distance it has to travel as much as possible, thus conserving power.

An interesting point to note is that in the naïve schematic discussed earlier, ρ is 1, which is the smallest value it can take. This is because the sensors treat PoIs independently, and travel in a straight line towards an individual PoI, making distance the same as displacement. This scheme, though energy efficient, increases the cost in terms of number of sensors. So we tend to search for techniques that result in a trade off between the two, i.e., reduce the number of required sensors drastically at the cost of spending some more power.

4.1 Centroid based Coverage Scheme

As discussed in the previous section, in the approach to club the sensors, a point needs to be chosen up to which the sensors are commonly propagated. As proposed in [14], that point is chosen to be the centroid of the given set of targets in the $2D$ plane. As proposed in the technique, a group of sensors are firstly propagated to the centroid, from where they spread out to cover each of the individual PoIs. This information can be fed into the sensors during the preprocessing stage. The number of required nodes is given by the following equations, assuming there are n targets:

$$\begin{aligned}
\overrightarrow{t_{mean}} &= sum(\overrightarrow{t_j})/n, \qquad \forall j \in [1, n] \\
num_{mean} &= \lceil \|\overrightarrow{t_{mean}}\|/r \rceil \\
num_j &= \lceil \|\overrightarrow{t_j} - \overrightarrow{t_{mean}}\|/r \rceil \\
num_{total} &= sum(num_j) + num_{mean} + 1
\end{aligned}$$

Once the number of sensors for the mean, as well as each target is determined. The deployment is easily carried out in two phases. The first phase is to propagate all the sensors to the centroid. The next phase is to propagate num_j number of sensors to target t_j using Algorithm 5 discussed in the previous chapter. The sensor which reaches the centroid last acts as a pseudo base station (as in it doesn't move anymore) for the remaining sensors.

The num_j sensors, which are allotted to a particular target, interact within themselves only. As an example, the sensors allotted to the same target can share a special code, which they send to their neighbors while communicating, and all messages in which the code doesn't match, can be neglected. As a result, all local decisions and updates are done only based on sensors inside the communication range, and allotted to the same target. This can be abstracted as a graph coloring.

4.1.1 Simulation

Figure 4.1 shows the result of the simulation of the above algorithm. The algorithm allots a total of 51 sensors for covering 5 PoIs located at $(24, 3)$, $(27, 8)$, $(4, 16)$, $(27, 29)$ and $(19, 29)$. The mean is located at $(20, 17)$. All of these sensors go towards the centroid and $8, 6, 9, 7$ and 6 sensors are respectively allotted to the targets from there on.

The communication radii of the sensors is taken as 2 while the sensing radii as 1. As seen from the figure, the algorithm achieves the coverage of all PoIs.

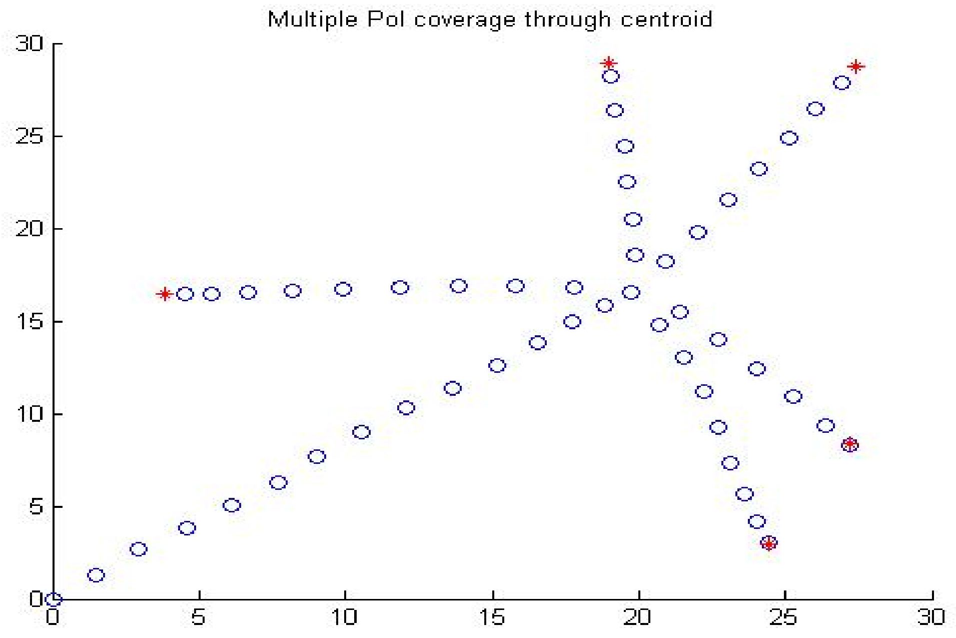

FIGURE 4.1: Simulation for multiple PoI coverage through Centroid scheme

4.1.2 Performance Analysis

Apparently, this technique reduces the number of required sensors, while increasing the distance needed to travel to cover a given sensor. Monte Carlo simulations are performed to determine the average decrease in number of sensors and increase in the distance. These values serve as a benchmark for the k-means clustering based scheme proposed in the next section. Figure 4.2 shows the comparison between the number of sensors required by the centroid scheme to that by the naïve one.

A linear regression performed on the data results in the straight line $y = 0.5x + 50$. The coefficients of this line are given by $w = (\phi^T\phi)^{-1}\phi^T Y$, where $\phi = [X \quad 1]$, X being the column of data for naïve scheme, and Y being the data for centroid scheme. 1 simple represents a column vector of all 1s. This shows that for deployment schemes where the number of required sensors is more than the order of hundred, the centroid scheme requires lesser sensors than the naïve one on average.

The Monte Carlo simulation for the maximum relative distance between the centroid and naïve scheme results in a mean factor of 2.6, which shows that the some sensor in the centroid scheme might have to travel up to 2.6 times what it has to in the naïve one on average. The proposed scheme in the next section aims to improve both these

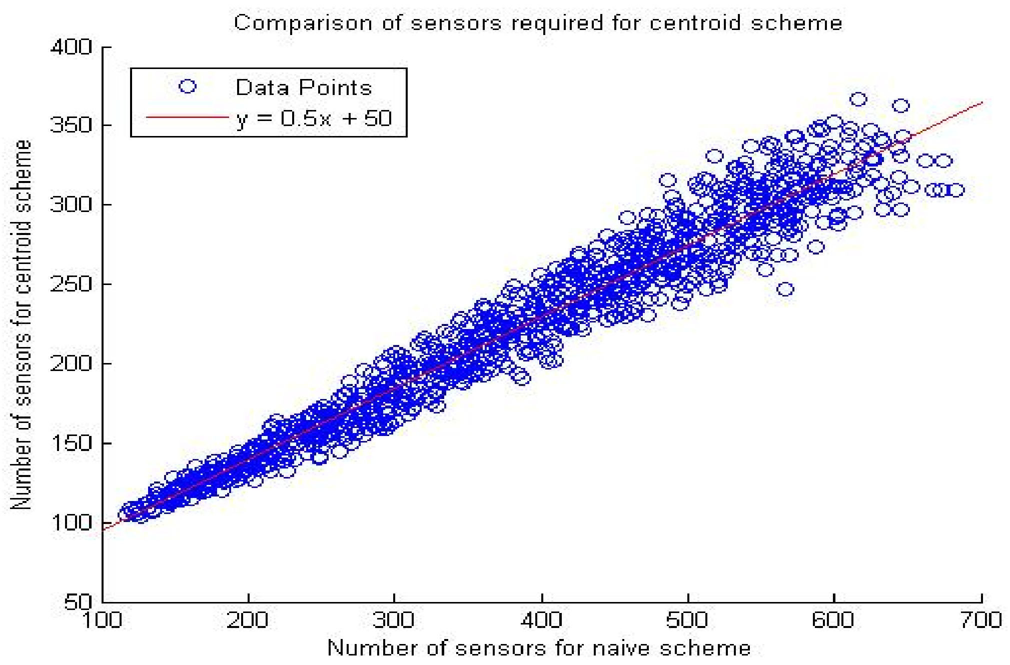

FIGURE 4.2: Number of sensors required by centroid scheme

parameters, i.e., reduce the number of sensors required, as well as the maximum relative distance needed to travel.

4.2 K-Means based Coverage Scheme

4.2.1 The K-Means Clustering Algorithm

A widely used technique for clustering points based on their Euclidean distance, is the K-Means clustering algorithm. It is an algorithm for unsupervised learning, and initializing with K means, recursively adjusts the locations of the means as to minimize the distance of any point from the mean of its cluster. Alternatively, it is said to minimize the sum of squared error, which is a popular choice for measuring the error. The algorithm is laid out below:

Since the initialization of the cluster means plays an important role on the result of the algorithm and bad initialization can lead to cluster formations without any nodes, a safe technique to counter this is to initialize the cluster means at k of the existing nodes. This results in at least one node being assigned to the cluster.

A major problem encountered in the K-Means clustering, is choosing the value of k, which basically affects the entire performance of the algorithm. Although there is no well defined procedure for choosing the value of k, usually, the aim is to select a k,

Algorithm 6 The K-Means Clustering Algorithm

```
$X = [x_1, x_2, x_3, ..., x_n]$ are the set of points
$k$ cluster means are chosen from the given points, denoted by $V$
for all $i \in X$ do
  if $\|x_i - v_j\|$ is minimum among all $j \in v$ then
    $cluster_i \leftarrow j$
  end if
end for
$cost \leftarrow 0$
for all $i \in X$ do
  for all $j \in V$ do
    $cost \leftarrow cost + \|x_i - v_{cluster_i}\|$
  end for
end for
while $cost_{new} - cost_{old} > 0$ do
  $cost_{old} \leftarrow cost$
  for all $i \in X$ do
    if $\|x_i - v_j\|$ is minimum among all $j \in v$ then
      $cluster_i \leftarrow j$
    end if
  end for
  for all $j \in V$ do
    $v_j \leftarrow 0$
    $count \leftarrow 0$
    for all $i \in X$ do
      if $cluster_i = j$ then
        $v_j \leftarrow v_j + x_i$
        $count \leftarrow count + 1$
      end if
      $v_j \leftarrow v_j / count$
    end for
  end for
  $cost_{new} \leftarrow 0$
  for all $i \in X$ do
    for all $j \in V$ do
      $cost_{new} \leftarrow cost_{new} + \|x_i - v_{cluster_i}\|$
    end for
  end for
end while
```

which minimizes the cost function. This is generally done by running the K-Means algorithm for a range of k, calculating the cost function for each of them and choosing the value of k that causes the largest dip in the cost function. A slight variation to this approach has been used in this application, as discussed below.

4.2.2 Coverage scheme

The selection of the clusters is done using the K-Means clustering, using the criteria for the selection of k as the number of sensors required. The value of k is chosen as to minimize the number the required sensors. The criteria is not set as the relative distance ρ, as this value can be reduced down to as low as 1, and will correspond to the naïve scheme. On the other hand, number of required sensors is large for naïve scheme, from where it falls, and again increases for centroid scheme. So, this algorithm aims to find that minimum value. The process for clustering the sensors is laid below:

Algorithm 7 Cluster Allocation

```
$sensors \leftarrow 0[\lceil n/3 \rceil]$
for $k = 1$ to $\lceil n/3 \rceil$ do
  $indicator_k \leftarrow 0[n \times k]$
  Run $K$-Means algorithm with current k
  for all $i = 1, ..., n$ do
    for all $j = 1, ..., k$ do
      $indicator_k[i, j] \leftarrow 1$
    end for
  end for
  for all $j = 1, ..., k$ do
    $sensors[k] \leftarrow sensors[k] + \lceil \|\overrightarrow{mean_j}\| / r \rceil$
    for all $i = 1, ..., n$ do
      if $indicator_k[i, j] = 1$ then
        $sensors[k] \leftarrow sensors[k] + \lceil \|\overrightarrow{sensor_i} - \overrightarrow{mean_j}\| / r \rceil + 1$
      end if
    end for
  end for
end for
for $j = 1, ..., k$ do
  if $sensors[j] = min(sensors)$ then
    return $indicator_j$
  end if
end for
```

Once the clusters are allocated to the targets, sensors are deployed, treating each cluster independently. This can be considered as a mixture of the previous two approaches, where a set of sensors are dedicated for each cluster, which propagate from the base station. Once the sensors reach their cluster mean, the sensor arriving last is declared as a pseudo base station in the sense that other sensors emanate from it, and go on to cover each of the targets in that cluster. Thus it also consists of two phases, like in the centroid scheme. Next, the simulation of the algorithm is done.

4.2.3 Simulation

Figure 4.3 shows the simulation of the K-Means based coverage scheme. The clusters are formed at $(8, 43), (26, 25)$ and $(42, 11)$. A total of 122 sensors are deployed, which propagate to the cluster means, and from there, to the individual targets. As seen from the figure, the algorithm achieves coverage of all PoIs.

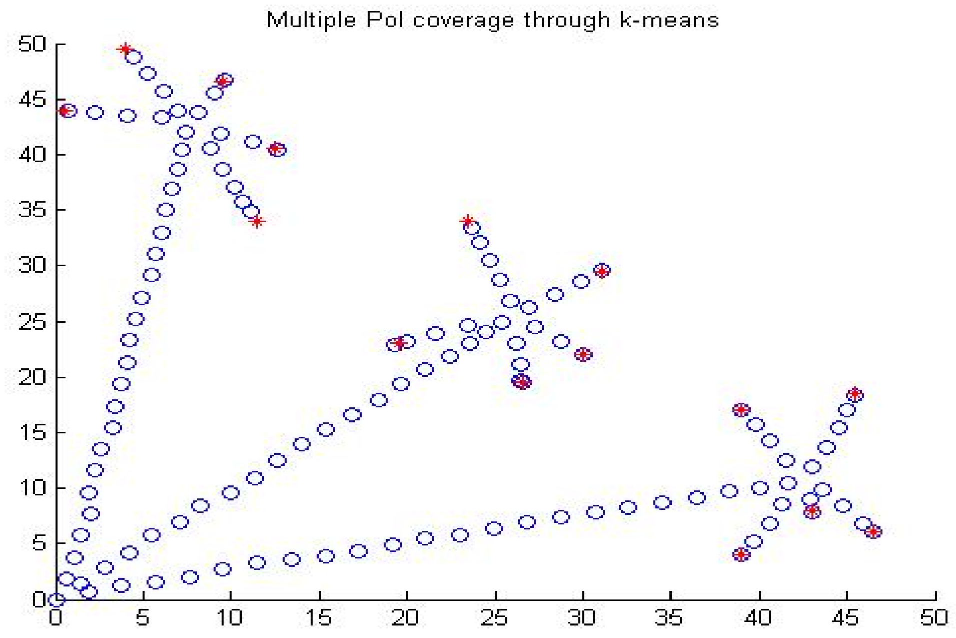

FIGURE 4.3: Simulation for multiple PoI coverage through K-Means scheme

4.2.4 Performance Analysis

Comparing to the performance of the centroid scheme, this method improves in both the aspects, it reduces the total number of sensors required, and the maximum distance that a sensor has to travel to cover a target is also lesser for this scheme on average compared to the centroid one. Monte Carlo simulations are performed to check both the aspects.

The result for the comparison of number of sensors required in the two schemes is shown in Figure 4.4. Performing a similar linear regression on this results in the line $y = 0.75x + 25$, which again means that when the number of sensors in the order of hundred, the K-Means scheme performs better than the centroid one on average, in terms of the number of sensors required.

The simulation result for comparison of the relative distance for both the schemes is shown in Figure 4.5. A regression is not performed as the data points diverge fast, and

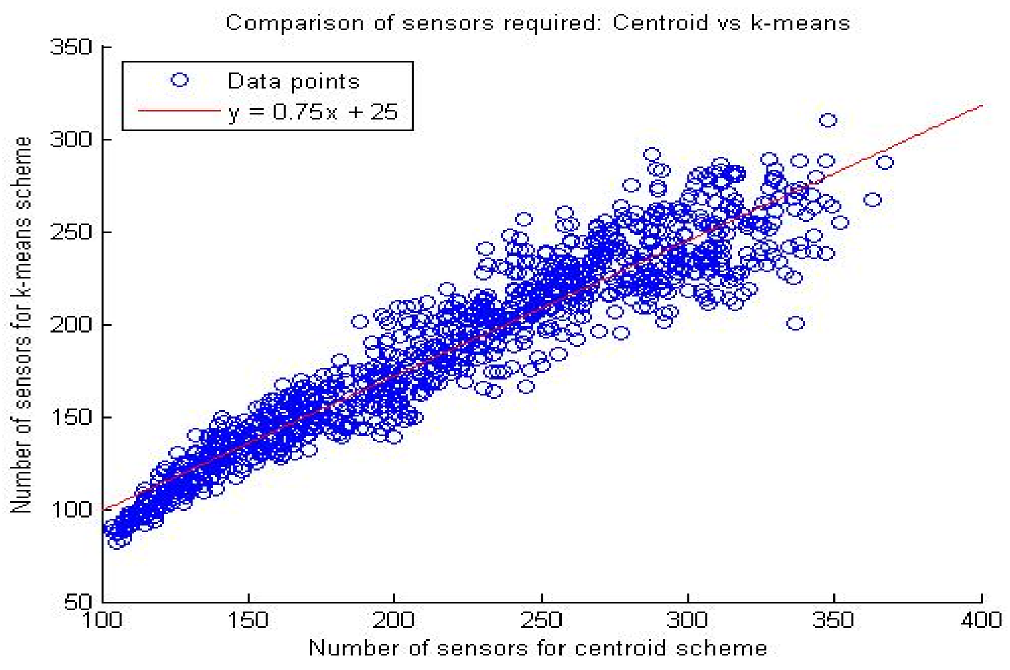

Figure 4.4: Number of sensors required by K-Means scheme

a straight line is not a good approximation, however, the red line which depicts the $y = x$ line always stays above the data points, indicates that the maximum distance that the sensors have to travel in the case of K-Means is almost always lesser than in the case of centroid scheme.

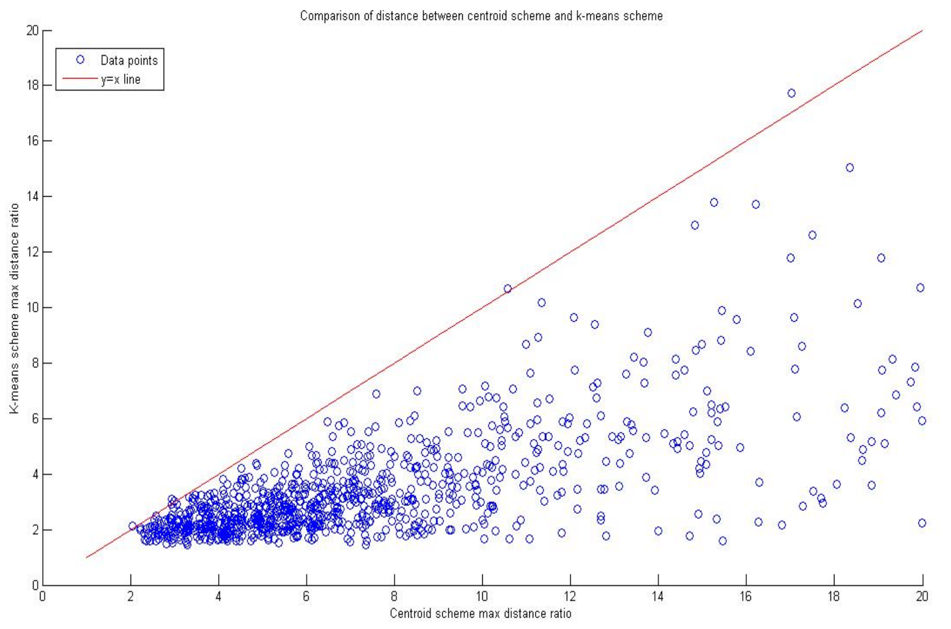

Figure 4.5: Relative distance for the K-Means scheme

Thus, for reasonably large fields, where the number of sensors is in the order of hundreds, the K-Means coverage schemes performs well both in the case of number of sensors required as well as the maximum distance a sensor has to travel in order to cover a target. It reduces the number of sensors required by approximately 3/4th, and the max relative distance is always less than that in the case of centroid scheme. So, it can be inferred that, the sensor allocation to the targets based on K-Means algorithm, is both cost-effective as well as energy-efficient.

Chapter 5

Conclusion

The report was organized in the following manner. It started with the description of the sensor network model. It defined it as a multi-agent system with an underlying graph structure. Coverage, which is the focus of this work, was next discussed as an important performance metric. Various notions of coverage were discussed after this, of which Area and Target Coverage are of particular importance to this work. Two major classes of algorithms, centralized and distributed were discussed, and it was argued that for the case of wireless sensor networks, distributed systems is a more suited choice as nodes in wireless sensor networks are always prone to failure, so replying on a central node is not a right choice.

Next, the problems of Area and Target Coverage were dealt with, and the conclusions from the work are presented below.

5.1 Conclusion from Area Coverage

The idea of potential game was applied to the sensor network, treating it as a multi-agent system. This allowed for design of local objective functions which led to a desired system-wide behavior. Within limits of marginal error, which occurred due to the uncertainty in the parameters involved, ϵ and α, the system achieved coverage.

Communication range of the sensors played a part in the speed of convergence; larger the range, faster the convergence. Also, at convergence, the value vector corresponded to the set that optimized the objective functions, and the estimation matrix resulted in all agents having correct estimates of all other nodes.

5.2 Conclusion from Target Coverage

The target coverage problem was specified as covering Points of Interest while maintaining connectivity. Using a Relative Neighborhood Graph, which is a connected graph and has a length stretch factor $\Omega(n)$, was a suitable choice. Further, its $O(n^2)$ computational complexity was reduced to $O(nlogn)$ by using the Urquhart Graph, which approximates the RNG with an error margin of 2%. Using this Urquhart Graph, the deployment process was carried out and coverage for a single PoI was achieved.

For the case of multiple PoIs, a K-Means based clustering was performed, and the clusters were chosen as to minimize the total number of sensors required. After this, each cluster was treated independently and covered with a dedicated set of sensors. This method turned out to be superior than considering the all PoIs as a single set by reducing the number of required sensors as well as the maximum distance to be traveled by a sensor to cover a PoI.

References

[1] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2):164–194, June 2005.

[2] Theodore Rappaport. *Wireless Communications: Principles and Practice.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2001.

[3] Bang Wang, Wei Wang, V. Srinivasan, and Kee Chaing Chua. Information coverage for wireless sensor networks. *IEEE Communications Letters*, 9(11):967–969, Nov 2005.

[4] Gaojun Fan and Shiyao Jin. Coverage problem in wireless sensor network: A survey. *Journal of Networks*, 5(9), 2010.

[5] Y. Yoon and Y. H. Kim. An efficient genetic algorithm for maximum coverage deployment in wireless sensor networks. *IEEE Transactions on Cybernetics*, 43 (5):1473–1483, Oct 2013.

[6] Jason R Marden, Jeff S Shamma, et al. Game theory and distributed control. *Handbook of game theory*, 4:861–899, 2012.

[7] Jason R. Marden. State based potential games. *Automatica*, 48(12):3075 – 3088, 2012.

[8] N. Li and J. R. Marden. Designing games for distributed optimization with a time varying communication graph. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 7764–7769, Dec 2012.

[9] Yingshu Li and My T. Thai. *Wireless Sensor Networks and Applications.* Springer US, 1st edition, 2008.

[10] Prosenjit Bose, Luc Devroye, William Evans, and David Kirkpatrick. *Theoretical Informatics: 5th Latin American Symposium Cancun, Mexico, April 3–6, 2002 Proceedings*, chapter On the Spanning Ratio of Gabriel Graphs and β-skeletons, pages 479–493. Springer Berlin Heidelberg, 2002.

[11] Godfried T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12:261–268, 1980.

[12] R.B. Urquhart. Algorithms for computation of relative neighbourhood graph. *Electronics Letters*, 16:556–557(1), July 1980.

[13] Godfried T. Toussaint. Comment: Algorithms for computing relative neighbourhood graph. *Electronics Letters*, 16(22):860–860, October 1980.

[14] Milan Erdelj, Tahiry Razafindralambo, and David Simplot-Ryl. Points of Interest Coverage with Connectivity Constraints using Wireless Mobile Sensors. In *Networking*, Valencia, Spain, May 2011.