Connected Mobile Sensors for Self-Deployment

by

Xueqian Wang

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

A Mobile Sensor Networks (MSNs) is normally made up of a number of mobile sensors which can be placed in a region of interest (ROI) by people. Sensors communicate with each other through wireless links to perform the distributed sensing ability for covering a region. Through a specific algorithm, the sensors move automatically, and finally the sensor network achieves a large sensing coverage. Sensing coverage can be established and promoted by different algorithms, and a good algorithm can lead sensors to form the largest possible sensing area without any sensing holes (areas that cannot be detected or monitored). The coverage of a sensor network is defined as the total area of interest covered, minus the area of the sensing holes.

We introduce a novel algorithm called the Spanning Tree-based Greedy-Rotation-Back (STGRB). The traditional Greedy-Rotation-Greedy (GRG) algorithm uses a point as a specific start so it's not appropriate for practical circumstances. So in the STGRB, we first use a spanning tree algorithm to get the gravity center (the sensor that connects the most sensors or is physically close to most of the other sensors) of the network. In this way we eliminate the first condition of selecting a sensor as a start point and also we conserve the energy of the sensors via letting them move a shorter distance.

Acknowledgements

I would like to acknowledge my sincerest gratitude to my supervisor, Prof. Dr. Amiya Nayak. Without his help and professional guidance, it would not have been possible for me to complete this thesis. I am so grateful to previous supervisor Prof. Dr. Ivan Stojmenovic for his support in preparation of this thesis. I would also like to thank Xu Li for helping me understand the issue more deeply. Finally, I would like to also thank my friends and particularly my family for their sustained understanding and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background Information

A Wireless Sensor Network (WSN) is a kind of distributed sensor network that consists of software, gateways, and a large number of distributed small and mobile sensors, which have limited and same communication range, computing, and sensing capabilities, and can move by themselves or on robots. Sensors and robots communicate with each other via wireless links in the region of the interest. The main purpose of a WSN is to generally supervise certain assets and collect data in a given region of interest (ROI) for various purposes, such as surveillance of forest fires, military usage, emergency response, and medical treatment. Due to the limited memory and computing capability of each sensor, sensors cannot implement complicated algorithms and protocols. Sensors and robots communicate with each other via wireless links in the region of the interest.

Two main methods of sensors deployment are currently popular in the WSN field. One is sensor self-deployment. With this method, sensors can deploy in the area by themselves, and there are many algorithms for self-deployment [1] [2] [3] [4] [5] [6]. The other one is carrier-based deployment. Sensors are placed by actors (such as robots) in the appropriate positions in an empty and unknown ROI. Generally, the former method randomly drops sensors using various machines (such as cars and planes) and the sensors moves according to the specific algorithm, so that the sensing coverage reaches a relative large value.

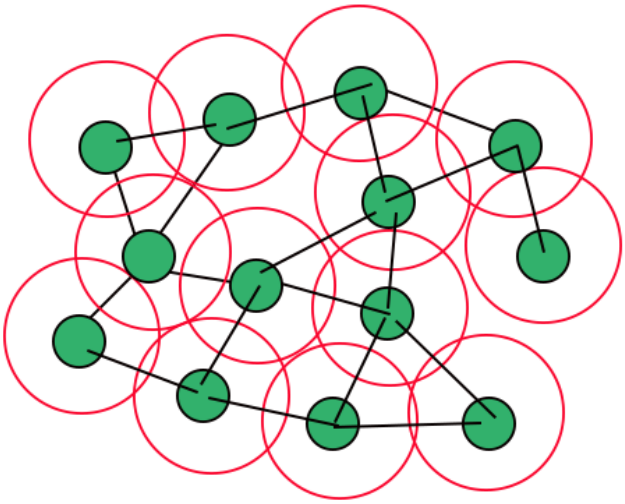


Figure 1.1: Wireless Sensor Network.

As shown in Figure 1.1, each sensor has a sensing range and a communication range. The red circle represents the sensing range and the segment between sensors shows the communication range. A sensor can communicate only to other sensors that are in the communcation range. Additionally, there are many overlapping areas of sensing coverage, so the question arises of how to minimize the overlapping areas and optimize the sensing coverage.

As soon as the network is initially formed, according to the algorithm, it should be stable in a desired period of time. In carrier-based deployment, actors not only move the sensors to the right place, but also have to replace the sensors in case some sensors break or there are sensing holes. By comparison, the self-deployed sensors can avoid the sensing holes by moving with a specific algorithm.

1.2 Problem Statement

1.2.1 Sensor Placement

A mobile sensor network (MSN) is an important concept for sensor self-deployment. In terms of the unpredictable broken sensor, limited network bandwidth, and large-scale networks, it is better that sensors make their decisions in an independent way. That's what we called a localized algorithm [24]. Generally, when a self-deployment algorithm is carried out comprehensively, in a 2D area, sensors with moving, sensing and communicating capabilities are dropped by airplanes randomly, which means the sensors are initially in the wrong places. Sensors don't know the specific positions they want to occupy. They just need to move to the right places by themselves according to a relatively simple algorithm, because sensors have limited memories and computing capability.

As every sensor knows its own unique ID (a number), the sensors can decide their order or relationships. Also each sensor includes GPS to track their positions and the distance between sensors accurately. Then, according to their IDs, the sensors in the MSN form a tree featuring father nodes (sensors) and son nodes (sensors). The network selects the gravity center as the point of interest (POI) so that the sensors can conserve more energy during their movement. Then, we use GRG algorithms to yield a network surrounding the POI with an equilateral triangle tessellation (TT) layout. With this layout, we can maximize the sensing coverage, such that no sensing hole exists in the network area when sensor separation is equal to $\sqrt{3}r_s$(sensing rage), and network connectivity is stable when r_c (communication range) $\geq \sqrt{3}$ r_s (sensing range) [7] [8] [9].

As each sensor is designed for low energy consumption at first, the algorithms should distribute the sensors in the area to conserve energy during the whole deployment. What we can do is to try our best to extend nodes (sensors) lifetime, extending the sensors' lifetime also reduces costs, including sensor exchange and sensor repair fees [10] [11]. In other words, we have to let the sensors move shorter distance to get larger sensing coverage.

1.2.2 Sensing Coverage

Once the POI (the gravity center of the network) has been selected, the sensors move asynchronously under the GRB algorithm. As long as the sensors get to the final positions where they belong, the mobile sensor network (MSN) should be stable. Meanwhile, sensing converage achieves its maximum value according to the principles of the algorithm. Even though we don't know the accurate finish time of the deployment, the operation time should be limited.

In the triangle tessellation layout, it looks like all the sensors move along a hexagon or the connection (one side of the triangle) between the neighbor hexagons. When sensors find there are no positions for them on the hexagon where they are, they will go (towards the counter direction of the POI) to the next hexagon to look for an available position. Eventually, sensors should occupy all the vertices if the number of the sensors is large enough. If there are more sensors than vertices, then of course some vertices will have more than one sensor.

1.3 Existing Solutions

One important issue of self-deployment is how the sensors move in the region when they are dropped randomly on the ground. This can also described as how the sensors communicate with each other under certain principles so that sensing coverage can be optimal. The existing solutions for the self-deployment have some apparent limitations, which will be listed briefly below.

Howard et al introduced the vector-based (or virtual force) approach [1], which is probably the best known sensor self-deployment method. The basic idea is: each node computes the virtual force (or movement vectors) caused by its neighbours using nodal relative position, and moves according to the summation of the forces (the total force). After a number of rounds of movement, the mobile sensor network reaches a stable status, which gives a nearly uniform node distribution and thus a nearly optimal coverage.

Heo and Varshney proposed an algorithm [12] which is based on the Voronoi diagram [13]. The Voronoi diagram is a computational geometric structure widely employed in different fields. Use of the Voronoi graph has been utilized in several algorithms [12] [14] [15]. N given nodes are used to divide the plane into n Voronoi regions and each region has exactly one node. The basic idea is: To minimize the uncovered area, sensors have to move and align their sensing range along their Voronoi polygon. However, sensing holes still exist when the network stays stable.

Bartolini et al. [4] wrote an algorithm named snap and spread self-deployment. This algorithm divides the layout into many hexagon grids and arranges sensors at the centers of the hexagon grid. Spontaneously, sensors start to establish hexagonal tiling through pulling and pushing nodes to hexagon centers. When sensors meet with each other, their tilling portions merge together. This algorithm is not a localized algorithm because a snapped sensor, in a pulling process, has to send messages to other hexagons before finding an unsnapped sensor.

The stochastic approach was first introduced by Mousavi et al. [3] and resembles a localized algorithm. According to this method, sensors are dropped randomly in an X $\times$ Y area and sensors, with limited random mobility, move around from a dense area to a sparse area based on their own knowledge. During the sensor deployment, time is divided into continuous equal periods. During a local period of time t, a node moves toward a position randomly and embraced by a moving rectangle uniformly. The network connectivity, sensing coverage maximization and eliminating sensor holes, cannot be guaranteed due to this algorithm's stochastic nature.

1.4 Motivations and Objectives

As many algorithms are not localized approaches, the cost (the number of messages) can be very high, to reduce costs we should try our best to minimize the number of the messages passed between sensors. Additionally, an effective algorithm should let the sensors move automatically without human manipulations. As long as the sensors have been

dropped from an airplane or other transport tools, they should move according to their own decisions.

As we stated previously, most algorithms are not strictly localized. Some algorithms cannot even guarantee the sensing coverage maximization, and some cannot guarantee sensing holes elimination. Even though the GRG algorithm is a strictly localized algorithm and it can guarantee no sensing hole, it cannot guarantee the sensing coverage maximization. So, we have to introduce a novel algorithm.

The GRG algorithm requires people to select a point of interest as input so that sensors can self-deploy around that point. We need to remove this precondition and let sensors absolutely self-deploy. Also, to avoid sensors moving a long distance to form a focused sensing coverage, and to avoid many collisions between sensors, we introduce a new localized algorithm named Spanning Tree-based Greedy-Rotation-Back (STGRB) which extends the GRG algorithm. Similarly, we introduce the concept of a gravity center, which means a sensor physically close to other sensors.

1.5 Assumptions

The Wireless Sensor Network (WSN) referred to in this thesis is totally composed of many homogeneous sensors with the same sensing radius and communication radius. The sensors are initially connected when they are dropped from an airplane or car. The relationship between sensing range and communication range is: r_c(communication radius) $\geq \sqrt{3}\ r_s$ (sensing radius). Here are the assumptions needed to present our idea in this thesis:

- Sensors know their own spatial coordinates by using some attached devices, such as a GPS device or other positioning services. As each sensor has to calculate the distance between its neighbouring sensors and itself, all the sensors have to know their own positions and their neighbours positions.
- Sensors know their 1-hop neighbours' location information and movement destination via lower-layer protocols [23]. As in our algorithm, sensors move strictly according to the TT layout, and each move is in tight relation with their neighbours' locations.

- Each sensor has a sensor ID. In our protocol, construction of a spanning tree network will be introduced in chapter 3. Based on the sensors' IDs, we can distinguish sensors from each other, and the root sensor can be determined by the largest sensor ID. After that we can use a BFS algorithm to construct the tree.

- All sensors in our protocol do not have fail conditions, which means sensors will always execute correctly during the simulation. As we just want to develop a better protocol, we do not consider fail conditions.

- Sensors in the simulation communicate to each other with omni-directional antennas. Their sensing range and communication range are Unit Disk Shape(UDS).

- As there may exist delays in propagating messages, we assume that the speed of the robotic sensors is much lower than the speed of sending messages.

1.6 Contributions

In this thesis, we introduce a novel algorithm named Spanning Tree-based Greedy-Rotation-Back (STGRB), which combines a sensor network spanning tree [30], gravity center and sensor movement approach (GRB) to allow maximum coverage, elimination of sensing holes, self-deployment, and low energy consumption. This algorithm gives enough "freedom" for the sensors to make their own decisions to move to form sensing coverage, and it improves the GRG algorithm for higher work performance in terms of energy consumption and area coverage. The spanning tree of the network is based on the concept that sensors are connected initially at first and therefore can find a gravity center that is physically close to other sensors in the region of interest. We see the gravity center as the point of interest.

The issue of sensor movement is the most important element in our thesis. We modify the GRG algorithm to make sure the sensing coverage is maximized according to the number of sensors. Once the number of sensors is not so large that they can cover the

whole region of interest, we have to make sure that the sensing area has been maximized without adding a sensing hole given the limited sensors.

1.7 Organization of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2, sensor self-deployment and carrier-based sensor deployment are reviewed. In Chapter 3, the STGRB approach is introduced in detail and the simulation consequences are shown in Chapter 4, which is followed by future work and the conclusion in Chapter 5.

Chapter 2

Literature Review

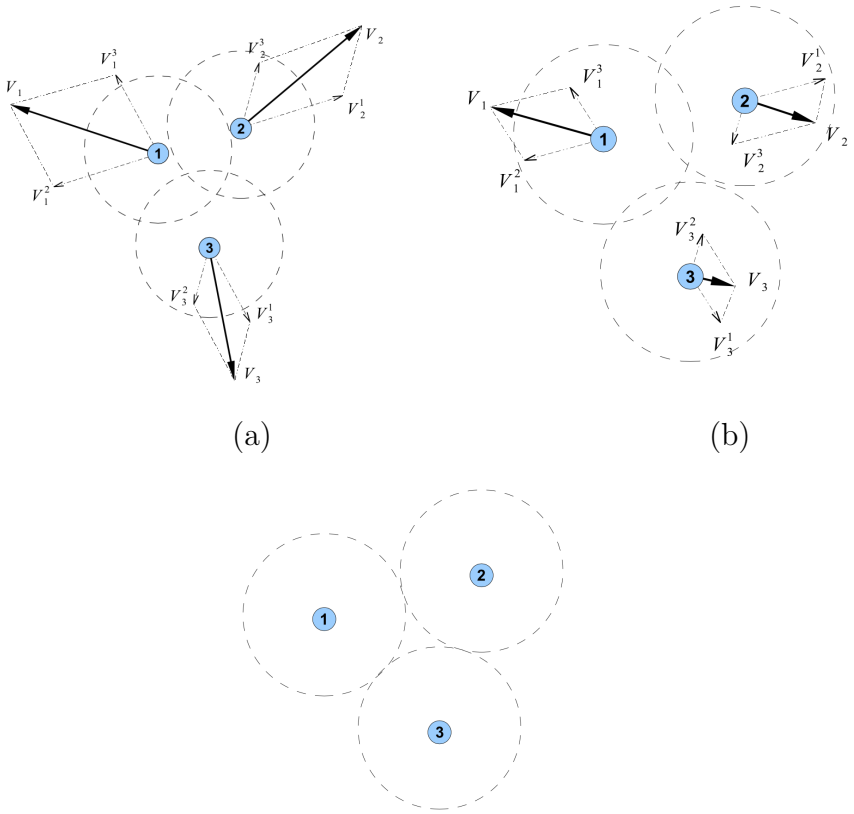
2.1 Sensor Self-Deployment

2.1.1 The Virtual Force Method

Nowadays, the virtual force (vector-based) approach has been applied in many fields with the aim of sensor self-deployment. The basic idea of the approach is not complex: each sensor moves according to a moving vector calculated by the relative forces of neighbours. Currently, there are many approaches inspired by this technique, but these implementations, despite different physical models such as molecules [12] and electro-magnetic particles [14], absolutely use a common philosophy as their theoretical core. Additionally, several variants have been proposed [20] [21] [22]. The goal of this method is to achieve uniform node distribution and thus approximate optimal sensing coverage.

Howard et al [1] introduced a potential field-based method based on the virtual force idea, and this approach is used to establish a Mobile Sensor Network (MSN) to maximize sensing coverage. During sensors movements, each sensor's direction is determined by its two neighbours. The total force of both neighbouring nodes pushes the sensor to move in a certain direction and this total force changes according to the neighbouring sensors' positions. Once each node reaches a balanced situation, the sensing coverage reaches its maximum value.

Figure 2.1 illustrates how three sensors move in an MSN. Initially, the three sensors are close to each other and the distance between two sensors is less than the sum of their sensing ranges. Due to that fact, their sensing ranges overlap and each of them feels the force generated by its two neighbouring nodes. The total virtual forces push the nodes to move apart from each other, maximizing sensing coverage. As the nodes move, it is apparent that the distance between node 1 and node 2 is becoming larger and larger. Meanwhile, the total forces become attractive and lead the sensors to move toward each other to minimize the area of sensing hole. Finally, the three sensing ranges touch each other without overlapping, and the sensing coverage reaches peak value.



(a) (b)

(c) final situation

Figure 2.1: Virtual Force Graph

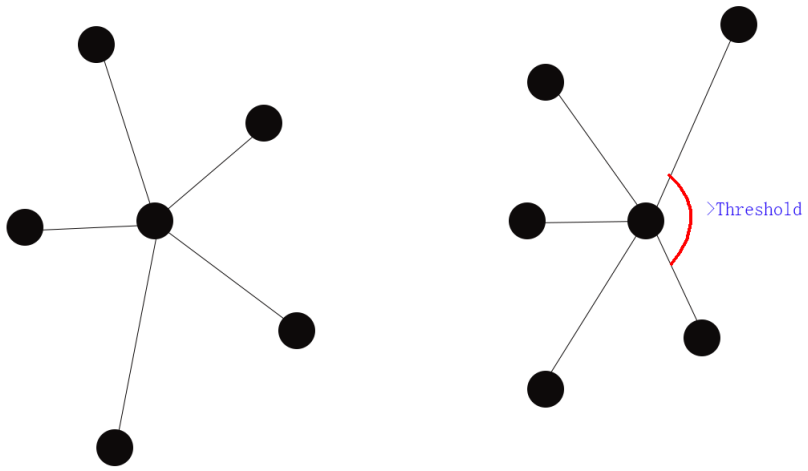
Garetto et al [17] proposed an novel algorithm in which sensors are dominated by events in the ROI. This event-driven self-deployment approach aims to form a regular triangle tessellation layout in the ROI. Based on events occurrence, sensors move to the active location and try to control the event that attracted the nodes. As soon as an event happens, sensors that are not close to the location move toward the event. After the event finishes, sensors return to their regular positions and resume the original sensing configuration.

A Vector-based approach is used in this algorithm because it is used to control sensor movement so that the node density around the event location is high enough when an event happens. There are three kinds of force: potential force from the detected event, exchange force from neighbouring nodes, and friction force. During the first step arrangement (sensor deployment), only exchange force and friction force have meaning when there is no event happening. In the algorithm, a node k has force on node i if and only if k is a neighbour of node i, and there is no other node k' meeting the condition $|k'i| < |ki| \wedge \angle k'ik < \frac{\pi}{6}$. Due to this, this condition ensures that no more than 6 neighbours can put their forces on node i in a given instant. As we previously stated, if the distance between sensors is less that $2r_s$, the exchange force is repulsive, otherwise, it will be attractive. Additionally, potential force also has two sides (attractive and repulsive). Potential force will push distant sensors towards an event location and pull in sensors that are close to event. Friction force always exists because it is used to stop sensor movement so that the MSN can be finally static and reaches a balanced status.

Poduri et al. [18] designed a virtual force-based method for controlling sensors' positions so that the network connectivity and sensing coverage can meet a desired level. How to arrange the sensors' positions is a crucial issue attempts to ensure that sensing coverage achieves peak value while meeting the network connectivity limit.

Based on the assumption that sensors are dropped densely in the ROI, a concept is introduced in this algorithm, defining that a node in the ROI has at least one neighbour in every θ sector of its communication range. In other words, the angle between any two neighbouring neighbours is less than θ. In the figures below, Figure 2.2 (a) meets the

conditions and is thus a valid NET graph. However in Figure 2.2 (b), since there is an angle larger than threshold θ, it is not a valid NET graph. When $\theta > \pi$, NET graphs have been proven to have an edge-connectivity of at least $\lfloor \frac{2\pi}{\theta} \rfloor$. As a result, based on the NET graphs, sensors can form a deployment that has the desired network connectivity and where there is only one parameter θ affecting the result. According to Menger's theorem [19], a graph is said to be k-edge-connected if k-edge independent paths exist between any two sensors in the graph.

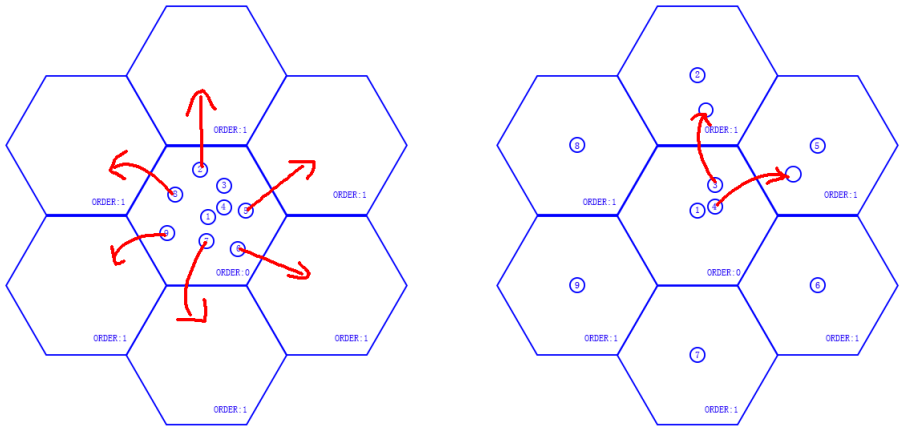


(a) (b)

Figure 2.2: Net Condition

2.1.2 The Point Coverage Approach

Bartolini et al [4] proposed a novel algorithm named snap and spread self-deployment algorithm, based on the assumption that region of interest (ROI) is bounded and the number of sensors is large enough to cover the whole ROI. In this approach, the ROI is divided into many hexagonal grids and the centers of the hexagons are the right positions for the sensors. Each hexagon's edge is equal to the radius of the sensing range. At the same time, nodes start to establish hexagon tiling over the ROI through setting the



(a) (b)

Figure 2.3: Snap Steps

position where it is currently as the center of the first hexagon. It then changes its status to snapped and sets its own order as 0.

As soon as a sensor finishes the steps that we just presented, it will study the status of its neighbors and then give ID numbers to the unsnapped neighbors in its hexagon. Each neighbor is then pushed to the center of a neighboring empty hexagon. When all neighbors get their destinations, they will update their orders as 1s. Figures 2.3 (a) and (b) below show the details of snap steps. In Figure 2.3, node 1 set its order as 0 and then pushes its neighbors (nodes 1-9) to its adjacent hexagons.

It is apparent that there still exist spare sensors in the node 1 hexagon. Therefore a spreading process will start, where the sensor pushes the spare sensors (those with smaller numbers and higher order) to neighboring hexagons. The closest node is selected when there are several nodes. Sensors in higher order hexagons will pull spare sensors by sending messages and then push them to higher hexagons if any exist. Through repeating these steps, redundant sensors are moved to higher order hexagons to expand the boundary of hexagonal tiling.

As this algorithm is a localized algorithm, all sensors run the algorithm independently

and many original hexagons may exist. However, these hexagons may not touch each other and overlapping areas exist. In this scenario, the earlier hexagon will "eat" the later hexagon and the snapped sensors in the absorbed hexagons will adjust their positions. Additionally, in terms of energy consumption, it is obviously that unsnapped sensors move much further than snapped ones to find proper positions. Snapped nodes and unsnapped ones may change roles during the movement to balance energy consumption. Based on this aim, one measure is used to balance energy that through controlling density. Once the sensor density in the target hexagon is lower than the threshold, we can control node density by limiting the number of snap and spread processes.

Li et al [23][24] introduced a novel concept, F-converage, connected with self-deployment problem. F-coverage makes sure that there is no sensing hole in a focused area around a point of interest (POI). Coverage radius is also first introduced in this algorithm, which is defined as the maximum radius of sensing hole-free disc coverage. In this algorithm, the author also presented two approaches Greedy Advance (GA) and Greedy-Rotation-Greedy (GRG). Both of these algorithms are strictly localized methods, which can help sensors move in the ROI according to their own wills.

The point of interest concept plays a vital role in this algorithm because all sensors move around to occupy the POI position. Assumptions are defined as follows. A number of sensors are dropped randomly from an airplane with $r_c \geq \sqrt{3}\, r_s$, where $r_c$is radius of the sensing range and r_s is the radius of communication range. ROI is designed in a Triangle Tessellation (TT) layout and sensors are aware of their specific positions as well as the POI position through GPS devices installed on them. As papers [7] [8] [9] stated, TT layout is used in this algorithm for sensing a hole-free and maximized coverage area when sensors are placed on vertices of the triangles. MSN also keeps sensors connected when those conditions are satisfied.

Greedy Advance (GA) is a method where sensors move greedily along the TT edge to get close to the POI. In other words, each node located at a vertex of a TT moves to another vertex, which is closer to the POI than its current location. According to the features of TT layout, there are at most two available vertices for movement, depending on where a

sensor is currently located. Sensors located on the corner vertices only have one available vertex for them to move towards the POI. In terms of controlling movement direction, the author introduces three rules in this algorithm. The first rule allows any sensors that are POI neighbors to move to the POI as long as POI is not occupied. The second rule arranges the order or priority to avoid two adjacent nodes colliding when they move to the same position simultaneously. The third rule, to avoid two non-neighbor sensors colliding, forbids potential movements of two non-neighbor nodes from their positions to the same vertex. Specifically, for two nodes located at $< i+1, j, 1 >$ and $< i+1, j-1, 0 >$, a node at vertex <i+1, j, 1> cannot move to vertex <i, j, 0>.

In Figure 2.4, if there are two sensors x and y (or y and z) moving to vertex k simultaneously, the node with higher order will get priority to move to k. When two nodes are not adjacent, like x and z, a movement from z to k is forbidden. However, due to rule 3, node 3 stops at the initial position of node 1 and doesn't go to position g even though g is empty.

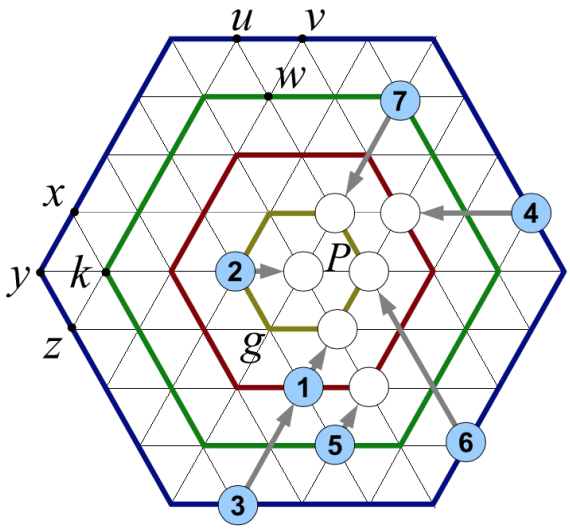


Figure 2.4: Greed Advance (GA)

To promote the GA algorithm, the author extends GA the algorithm to GRG (Greedy-Rotation-Greedy) by adding a new type of movement, rotation. Rotation movement is triggered when a sensor's movement is blocked. The rotation direction is pre-determined,

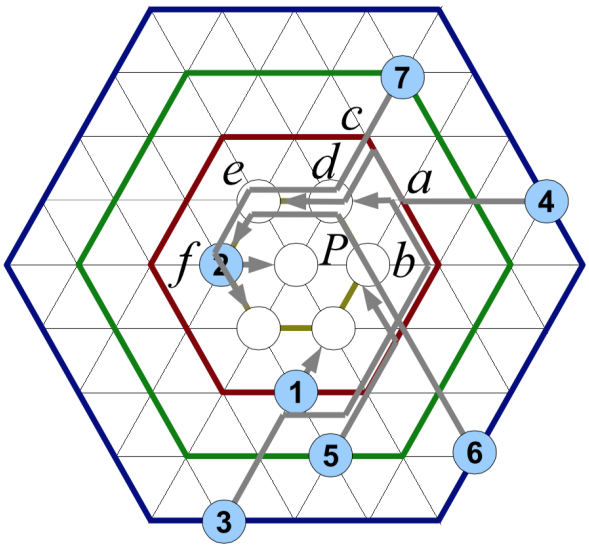


Figure 2.5: Greedy Rotation Greedy (GRG)

and is clock-wise or counter-clockwise along the hexagon edges. Once there is an available vertex for its greedy advance movement or it goes back to its original position, a rotating sensor will stop rotating. Considering that rotation behavior may block a greedy advance movement, there is another rule for avoiding that circumstance. As soon as a rotating sensor located in i layer finds that its neighbor located in i+1 layer wants to greedily move to its destination vertex, it suspends its rotation behavior. Figure 2.5 illustrates how sensors move according to the GRG algorithm. More specifically, the grey arrow lines are the paths that sensors move along according to the GRG algorithm. Focus on nodes 2, 4 and 6. Node 2 moves to its final destination by implementing only one step. Nodes 4 and 6 will travel a relatively longer distance to get their final destinations. As we can see in the figure, node 4 moves to vertex a via greedy advance movement and then it finds vertex d is occupied by node 7. As soon as it rotates to vertex c, mode 6 reaches at vertex b. Node 4 notices that vertex d is available now and node 6 wants to rotate, so a collision happens at vertex d. According to the rules of GRG, node 6 has higher priority to take vertex d as its next deployment step. So node 6 keeps rotating and gets to its final position f, while node 4 passes d after node 6 and then reaches at its eventual position e after node 6 leaves e for f.

GA has a shorter convergence time than GRG and it has been verified that both of them can terminate in finite time. To improve the GRG algorithm, the author altered the GRG algorithm based on the assumption that an obstacle exists in the ROI. GRG/OA is presented under that consideration and is used to let sensors penetrate that obstacle.

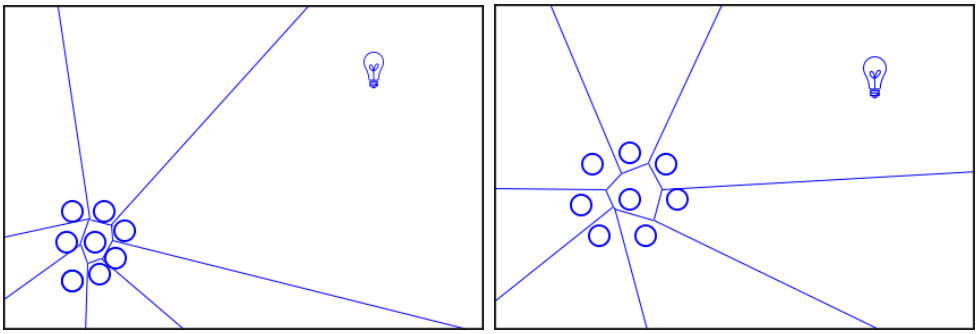
2.1.3 The Voronoi-based Approach

As a computational geography structure, the Voronoi diagram [13] has been used in many different fields. More importantly, use of the Voronoi diagram has been considered in the self-deployment field, as presented in several papers [12] [14] [15]. The diagram partitions a 2D plane into n Voronoi regions by using n given nodes p_1, p_2,..., p_n.

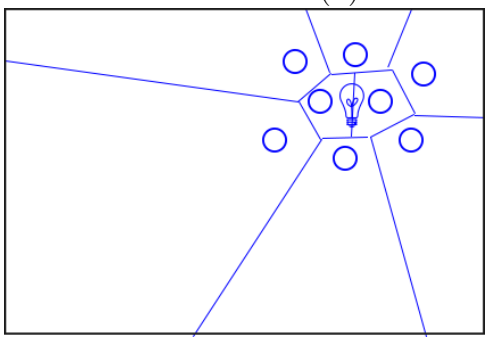
The basic idea of Voronoi-based self-deployment is not complex: through touching sensing range with Voronoi diagrams, sensors move around to minimize their uncovered area. However, approaches to sensing range touching differ from paper to paper. Generally, after a number of rounds, the Voronoi-based method always terminates when there is no more gain (sensing coverage gain in [14] and utility gain in [12]). In [14], to get close to the furthest Voronoi vertex, a sensor always travels a distance that is equal to half of the communication range. In [12], based on a computation of the sensor's effective area and its lifetime, a sensor moves to the position where it can absolutely use its energy in an effective area. In [15], each sensor moves to the centroid of its Voronoi diagram. The method introduced in [15] will be presented briefly below.

Cortes et al. focused their research on coverage control in MSN. There are two main elements of the coverage ability: the desired utility and a formula of its location. To simplify the method, just covering a target and maximizing sensing coverage are considered in this circumstance. Assume that each sensor knows its own location and its Voronoi neighbors' positions. How to maximize nodes' detection probability by controlling their movements is a crucial issue for this algorithm.

Mobile sensors update their Voronoi polygons while the sensors are moving. All sensors travel for a long distance to reach the final locations where their ability of surveillance can be optimized. With a Gaussian density function, it is easy to compute Voronoi polygons'



(a) Initial distribution (b) Transient distribution



(c) Final distribution

Figure 2.6: The Voronoi Approach

centroids, which represents declined ability of surveillance for a location that is further from the target. Robots' (or mobile sensors') positions are also decided by their Voronoi neighbors' positions. With the movements of centroids of neighboring Voronoi polygons, mobile nodes move forward to the centroids rather than to the geographical centers of Voronoi polygons. The reason is that weighted centroids of Voronoi polygons are willing to be closer to the source than geographical centers. In Figure 2.6 (a) below, the initial positions of the robots and their transient distribution are shown in the graph, respectively. The final situation of self-deployment is illustrated in Figure 2.6 (c).

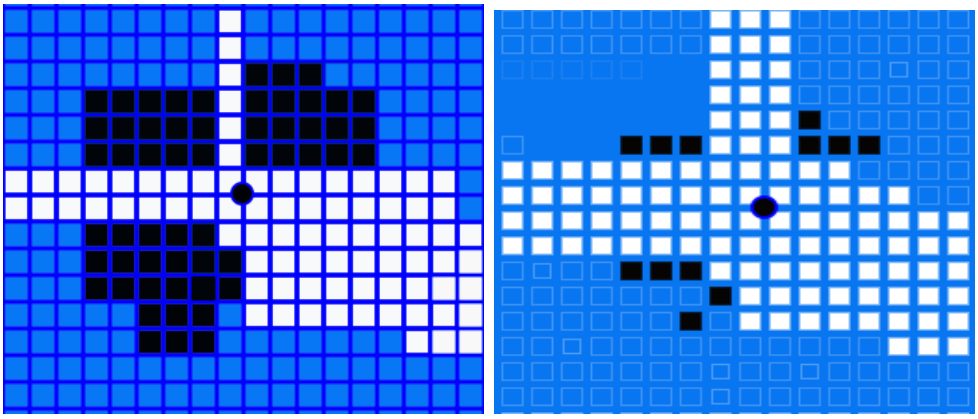
For each weighted centroid, its neighboring robots' coordinates can also decides its location. Each robot moves to the centroid located in its corresponding Voronoi polygon. Additionally, in this algorithm, Voronoi graphs are constructed many times so that sensors' movements can be presented. As a result, there is a large message cost of this method.

2.1.4 The Incremental Approach

Howard et al. [26] proposed a novel self-deployment method named the incremental approach. This algorithm aims to establish an MSN with full visible area maximization. Based on the assumption that each sensor has vision ability, each time only one sensor is deployed with an additional visibility constraint. In other words, sensors must be in at least one other sensor's sight.

There is a central controller in this algorithm for iterations. Initially, assume that there is only one sensor to be deployed. Repeating this step, the controller deploys only one sensor per iteration until all spare sensors are placed at the right positions. The central controller selects sensors' positions depending on where previous sensors have been deployed. Firstly, the central controller will construct an occupancy grid. In the grid, there are three kinds of cells: free cells, occupied cells, and unknown cells. The free cell is defined as one where there is no obstacle in the cell. A cell is occupied if the cell contains obstacles, and unknown otherwise. As soon as the occupancy grid is transformed to configuration grid by the central controller, in the configuration grid, a cell is said to be free if and only if all neighbors (cells that are in a certain range of the cell) are free. And a cell is said to be occupied if one or more cells in the range are occupied. Otherwise, a cell is unknown. After the configuration grid is constructed, it will be converted to a reachability grid. If there are some consecutive free cells between a cell and a certain deployed node, then this cell is said to be reachable. Otherwise, the cell is said to be unreachable. After the reachability grid is established, the central controller decides the next position for node to be deployed, where this deployed node can reduce unknown cells and the overlapping sensing area. Figure 2.7 shows all three phase graphs generated by only one node [26]. The three figures below are the occupancy graph in Figure 2.7 (a), configuration graph in Figure 2.7 (b), and reachability graph in Figure 2.7 (c).

As soon as the position is decided, using the Dijkstra Algorithm [25], the shortest path between the current node and previously deployed node is detected. To balance sensors' energy consumption and avoid blocks between sensors, sensors' movement is implemented by a sequential shifted behavior along the path. After the sensors have gathered vari-



(a) Configuration (b) Occupancy

(c) Reachability

Figure 2.7: The Incremental Approach

ous information, the central controller will get information transformed from the sensors and then construct the occupancy grid again. Even though this algorithm has a relatively shorter time convergence, it is not a strictly localized self-deployment algorithm. Additionally, it will be pricy for the bandwidth and energy consumption because of its centralized method.

2.1.5 One step Deployment

Mousavi et al[30] presented a distributed One Step Deployment(OSD) algorithm. In this algorithm, the layout is a grid layout and the relationship between the range of communication and sensing range is: $r_C \geq \sqrt{2} r_S$. The aim of this method is that the full region will be covered if every grid point in the area is occupied by a sensor.

In OSD, a spanning tree will be constructed based on a selected root sensor and a convergence process is then executed so that each node in the spanning tree knows its number of sub-nodes. After that, the root sensor will select gird point (0,0) as its own deployment aim position and assign each of its sub-trees a sun-area in the region of interest. Therefore, a recursive area assignment process starts. At the end of this process, each sensor in the region knows its destination and moves there by only one step to form a sensing covereage. So the region of interest will be fully covered and there is no sensing hole in it.

2.1.6 Comparison of Approaches

Table 2.1 below compares between different approaches and their advantages and disadvantages.

2.2 Carrier-based Sensor Deployment

Generally, in some circumstances, sensors are static and their movements are assisted by actuators. Some sensors are placed in the ROI by more than one robot. Robots are always assigned with a few specific routes for placing sensors and keeping the network connected.

Table 2.1: Comparison of Approaches

Approaches	Types	Advantage and Disadvantage
Virtual Forced	Distributed or Localized	Advantage: Localized method Disadvantage: Existing sensing hole
Point Coverage		Advantage: Strictly localized method Disadvantage: Needs TT layout
Voronoi Based		Advantage: Shorter time convergence Disadvantage: Not strictly localized
One step Deployment		Advantage: Energy conservation. Disadvantage: It requires all sensors connection at the beginning.
Incremental Approach	Centralized Approach	Advantage: Shorter time convergence Disadvantage: Not a localized approach

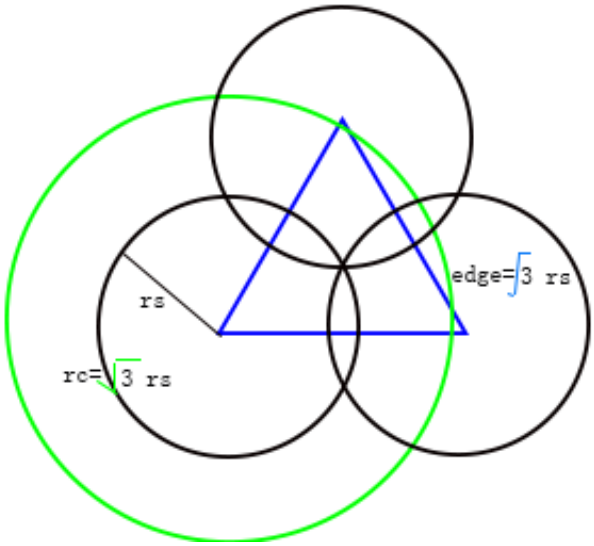


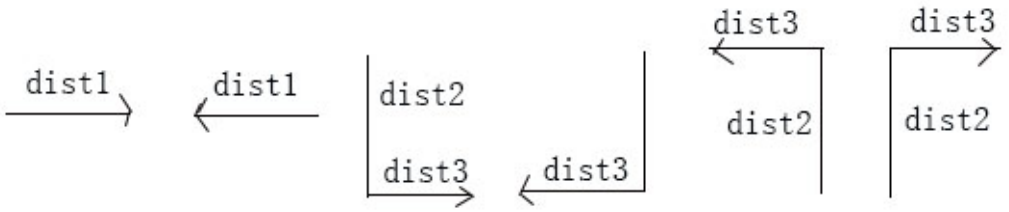
Figure 2.8: Distribution with TT layout

Even though self-deployment nowadays draws a lot of attentions, deploying sensors by actuators is still an interesting topic in the Wireless Sensor Network field. There are three main algorithms for Carrier-based Sensor Deployment: the Obstacle-Resistant Robot Deployment approach (ORRD) [27], the Least Recently Visited approach (LRV) [28], and the Back-Tracking Deployment approach (BTD) [29]. We will address all three algorithms briefly in this section.

2.2.1 The Obstacle-Resistant Robot Deployment Approach (ORRD)

Chang et al. [27] proposed the Obstacle-Resistant Robot Deployment (ORRD) approach. This approach is extended from a previous method called the Obstacle-free robot deployment approach (OFRD) introduced by Chang et al. [30], which resolves the problem of sensor deployment by only one robot.

The ROI is distributed into a Triangle Tessellation (TT) layout and robots move along the edges between vertices, which is similar to the GRG algorithm that we introduced previously. The TT layout is illustrated in Figure 2.8. The sensing range of each sensor is represented by a black circle.



Type 1 Type 2 Type 3 Type 4 Type 5 Type 6

(a) (b) (c) (d) (e) (f)

dist1=$\sqrt{3}r_S$, dist2=$\frac{1}{2}r_S$, dist3=$\frac{\sqrt{3}}{2}r_S$

Figure 2.9: Six Types of Directions

As we can see from the Figure 2.8, each sensor needs to send a message or communicate with neighboring sensors, so its communication radii is $\sqrt{3}$ r_S. r_S and r_C are defined as the range of sensing and range of communication, respectively. So to ensure the communication of sensors, the relationship between the communication radii and sensing radii must satisfy the condition below:

$$r_C \geq \sqrt{3}r_S$$

In this method, a robot only can move in four geographical directions of each vertex: left, down, right and up. That's the difference between this algorithm and the point coverage approach. For instance, if a robot wants to move from the left vertex to the top vertex in Figure 2.8, it does not travel along the edge of the triangle. It will move towards the up direction in distance $\frac{1}{2}$ r_S and then towards the right direction in distance $\frac{\sqrt{3}}{2}r_S$. When the robot is arranged by the controller to place sensors, it will move along the horizontal line (left or right) to place a sensor every $\sqrt{3}r_S$ distance until it reaches the boundary of the ROI or obstacle. Once that situation happens, it moves $\frac{\sqrt{3}}{2}r_S$ down to the next horizontal line for the next iteration. According to the method of actuator movement, there are six types of movement of robot. The six types of movement are shown below.

At the beginning of the movement, there are two directions for the actuator, which are East and West. When the robot is in one of the two situations, it has six directions to

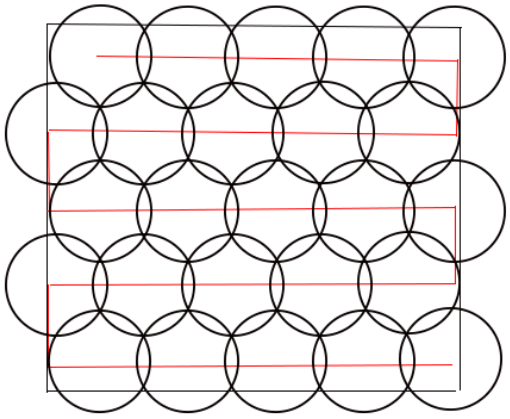


Figure 2.10: The ORRD Approach

choose from. Specifically, the six directions can be divided into a set of Checking Directions and a set of Prefered Directions. In other words, each set has three directions. The two sets have different functions: Checking Directions are used to check if any sensing holes exist, and the robot's next movement direction is decided by the Preferred Directions. Before every next movement step, robot will search three Checking Directions to avoid sensing hole existence. If the feedback from the Checking Directions is yes, that means there are some sensing holes, so the robot will move to the sensing hole's position first. Otherwise, actuator moves to the next location according to one of the three Preferred Directions. In Figure 2.10, the whole process of actuator movement is shown in the graph. The red line indicates the path that the robot has traveled. The disk shapes are the sensing ranges of all the sensors dropped by the robot. Additionally, the black lines are the boundaries of the ROI. In other words, the square area is the ROI. For the assumption at the beginning, the robot cannot move out of the the ROI and it will choose a new direction as long as it reaches boundary of the ROI or obstacle.

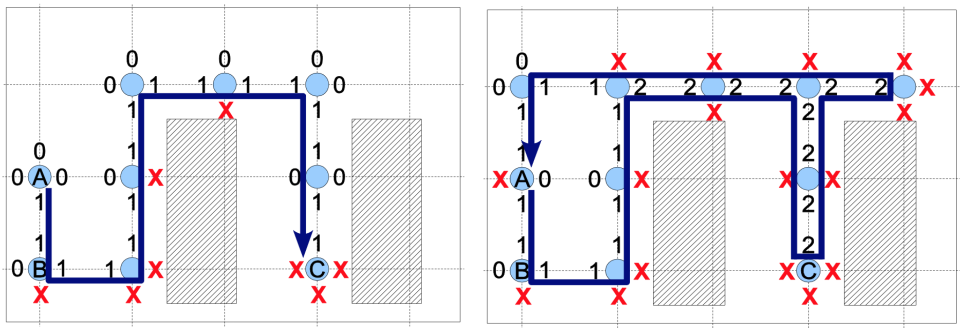
Obstacle-resistant robot deployment (ORRD) is an advanced application of the OFRD approach. The author puts his key points on addressing boundary and obstacle problems. Even though this method has a shorter time convergence, it is not clear that we do not know the accurate termination time in both OFRD and ORRD approaches. Additionally, this method cannot guarantee sensing coverage maximization without sensing holes.

2.2.2 The Least Recently Visited Approach (LRV)

The Least Recently Visited Approach (LRV) approach was introduced by Batalin and Sukhatme [28], and is based on a single-actuator scenario. In this algorithm, each sensor has the same sensing and communication radii and the actuator is guided by the recommendations of previously deployed sensors. Originally, the robot places the sensor at the location where it is. Each deployed sensor has directions that the actuator can move along to leave from it. Additionally, each direction has a weight representing the number of times that the robot has moved along it. The directions can be pre-defined or can be the edges of a graph structure. Initially, each direction weight is set to 0 and direction weight increases when the actuator leaves from a signal in that direction.

During the actuator movement, when a robot is in the sensor's communication rage, the sensor will submit its own least recently visited direction to the actuator. At the beginning of the process, directions are ranked so that only one direction is recommended to the actuator in the event of a tie. As soon as the direction is submitted to the robot, the robot will move along the direction and drop a new sensor. Once the actuator encounters an obstacle or the boundary of the ROI, it will send a message to the sensor asking for another direction to move along. This action can be easily implemented by communication between the sensor and the actuator. Therefore, the robot will stay at the position that it just achieved for a while because it has to wait for the recommendation submitted by the sensor. As long as the robot leaves from the sensor, the weight of the selected direction that robot moves along will be added. During the robot movement, if there is no message from the sensor, which means the robot is out of the communication range of the sensor, the actuator will drop a new sensor in the current location in the ROI.

Figure 2.11 shows the whole process of sensor deployment. Initially, there are four preordered directions which are South, East, North and West. The track of the robot movement is indicated by a blue line and the numbers around each sensor are direction weights for that sensor. The red crosses are the sign of denied direction, which means the direction is blocked. Originally, the actuator is at location A and it drops a sensor at that location. Due to the preordered directions, the actuator moves to the South side and



(a) (b)

Figure 2.11: The Least Recently Visited Approach

achieves at location B. As soon as it reaches location B, it moves out of the communication range and drops a new sensor at the location. The current sensor immediately recommends the South direction to the actuator, and then this direction is denied because of the boundary of the ROI. The robot asks for another direction from the sensor and, according to the rank of directions, the sensor submits the East direction to the actuator. Therefore robot selects this direction to move along and, after several iterations, it reaches location C, as shown in Figure 2.11 (a). After that, the robot has to go back along the previous route with the direction North, because all the other directions are blocked. Using the previously deployed sensors' recommendations, it travels back to its original position and drops new sensors at the right locations, as shown in Figure 2.11 (b).

It is apparent that the Least Recently Visited approach is a strictly localized method. However, it does not specify under what conditions the algorithm terminates. In other words, if nobody gives an outside limit to the robot, it will always run to place new sensors in the ROI. As there is no GPS device on the robot, the robot cannot have a global view of the ROI and it will keep receiving messages from the neighboring sensors and running in the ROI.

2.2.3 The Back-Tracking Deployment Approach (BTD)

Li et al. [29] introduced a localized algorithm named the Back-Tracking Sensor Deployment Approach(BTD). It is based on the assumption that all robots have knowledge of their locations and have ability to detect boundaries, obstacles, and previously deployed sensors. Like LRV algorithms, there are four pre-defined directions in different areas with different ranks. According to the rank of directions, each robot moves along the highest ranked direction, and it may also move along the second highest direction if the first highest direction is blocked. During the movement, the actuator drops sensors at the right locations (according to the layout, i.e., square grid or TT grid). It has been verified that the BTD algorithm terminates within a finite time and can form desired sensing coverage.

In a grid network topology, a vertex is said to be adjacent to another vertex if and only if there is an edge between them. An available vertex is a spot that is not occupied. All of the sensors have two colors, which are white and black. If a sensor's color is white, that means it has a neighboring available vertex. If all the neighboring vertices of a sensor are occupied by sensors, the sensor's color will be black. Additionally, there are two relationships between sensors. The predecessor of a sensor is defined as the sensor that was deployed before it by the same actuator. A node is said to be a successor of a sensor if the sensor was placed right after it. Each node in the ROI has two pointers: one is the forward pointer and the other one is the backward pointer. The forward pointer represents the arrow line from a sensor to the sensor's successor. Similarly, the backward pointer points to the location of predecessor if the predecessor's color is white. Also it can point to the predecessor whose backward pointer points elesewhere. Forward or backward pointers can be set to nil if a sensor does not have a successor or predecessor.

The numbers of sensors dropped by the same actuator increase over the course of a deployment. Sensors can judge if they were dropped by the same robot according to the robot ID. Different robots have distinct IDs and they will send their IDs to the sensors they drop. In a square grid layout, if the four directions are blocked, then we say the actuator reaches a dead-end. Once this situation happens, the robot will execute a backtracking process and move along the backward pointers chain from the position where it is now

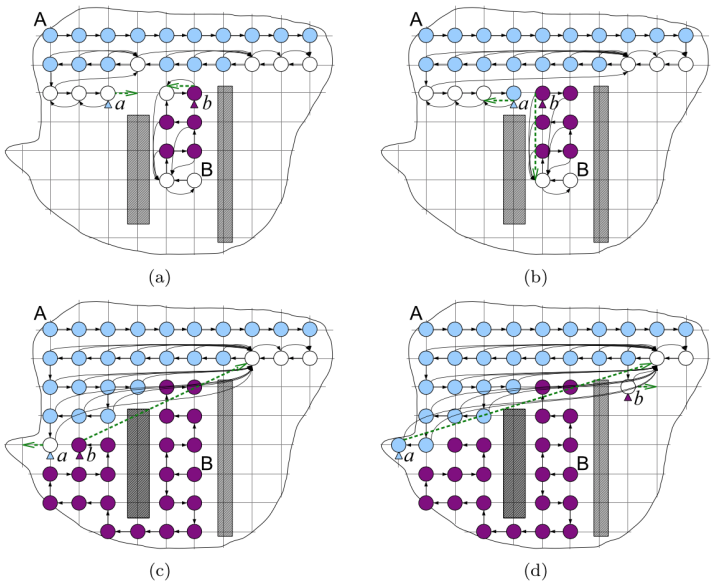


Figure 2.12: Back-Tracking Deployment

to a previously deployed white sensor. As soon as it finds the target, it will resume the process for exploring the ROI. An actor is said to serve a white sensor if it executes backtracking process for reaching the white sensor. For a sensor, if the number of its neighboring available spots is equal to the number of serving robots, then we say the sensor is fully served. Generally, robots will send a message to white sensor before it starts to serve the sensor. Only after its request is accepted, can it take any actions for serving the white sensor. Due to the uncertain nature of network, when a robot drops a sensor, this behavior may cause neighboring sensors to change color to black. As there exists delay of propagation, an actuator may move to a black node from a dead-end. Once this situation happens, the robots reach another dead-end. Figure 2.12 shows the four dead-end situations.

A bad situation is that more than one adjacent sensors fail in a backward pointer chain. In this case, a sensing hole may exist in the ROI and the robot will move into the sensing hole area to fix it. If there are some white sensors around the sensing hole, the actuator will move along backward pointers chain and place sensors on the right positions after the hole is fixed.

Chapter 3

The Spanning Tree-Based Greedy-Rotation-Back (STGRB) Method

In this chapter, we will introduce a new algorithm named STGRB. This algorithm contains two parts: constructing a spanning tree of the wireless sensor network connectivity graph, and a Greedy-Rotation-Back part. We will describe the details of both parts. In the first part, sensors will form a spanning tree of network connectivity according to their own attributes (ID number and coordinates). The second part will illustrate the triangle tessellation layout of the region of interest and details of protocols of sensor self-deployment. The whole process of STGRB algorithm is presented in Algorithm 1 below.

In Section 3.1, we will introduce the first phase: constructing spanning tree and selecting the gravity center of network. After that, the second phase running the GRB algorithm will be described in Section 3.2.

Algorithm 1 STGRB algorithm

Drop sensors in region randomly
implement Back Process
if Sensors are located on different vertices **then**
    Construct spanning tree based on root sensor
    Select gravity center based on the spanning tree
end if
if the spanning tree is a single-centroid spanning tree **then**
    run GRB algorithm based on the gravity center.
else
    randomly select one of the two centroids as gravity center
    run GRB algorithm based on the gravity center
end if

3.1 The Spanning Tree

3.1.1 Finding the Root Sensor

First of all, as we stated previously, the sensors dropped from an airplane are connected with each other and no sensor is separated from the majority of the sensors. Under this circumstance, we want to form a spanning tree of the network connectivity graph, so that we can get the gravity center (sensor). According to the ID, the sensors have to select the root sensor first, whose ID is the largest number among the sensors.

Each sensor has attributes (sensing rage, communication range, and ID) with them and they know where exactly they are. And through passing messages to others, each node in the network knows other nodes' positions (coordinates). When a sensor starts to work, considering itself as the root of the spanning tree, it sends the information (Root ID, distance=0) to its neighbours where the Root ID is its own unique identifier and the distance represents its assumed distance to the currently selected root. At first, each node sends its own ID as the Root ID to its neighbours and the distance between itself and root sensor is 0. By sending information to neighbors, all nodes thus know their neighbors'

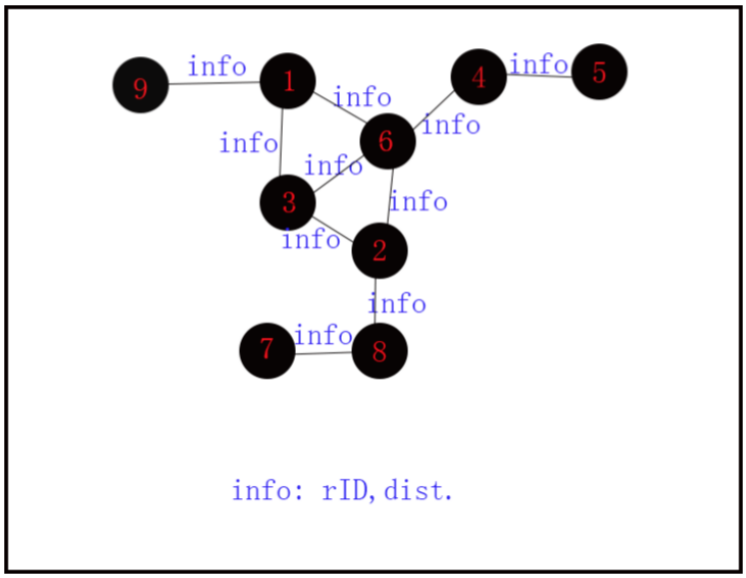


Figure 3.1: Finding Root Sensor 1

information and the current root sensor. The information keeps changing due to updated information. This phase will finish when all sensors' neighbors' root IDs are the same. After that, the root sensor is determined. Additionally,each node knows the distance between it and the root sensor.

At the beginning of the algorithm, the nodes send their information (root ID, dist=0) to their neighbours, and then each node receives messages from its neighbours. When node a receives the message (root ID, dist) from node b, it will updates its assumed root and parent as follows. If the root ID is greater than the current root it will be selected as the new root. After that, node a will update its parent as node b, and will also send messages (root ID, dist+1) to its own neighbors. Otherwise, node a sends an ignore message to node b. It is apparent that when a node receives ignore messages from all of the neighboring sensors it will define itself as the leaf of the spanning tree. In Figure 3.1, there are nine sensors and each sensor has its own identifier (ID). Sensors send information to each other, and the information includes the root ID and the distance between itself and the root sensor.

In Figure 3.2 sensor 9 has received a message from node 1. By comparing the rID

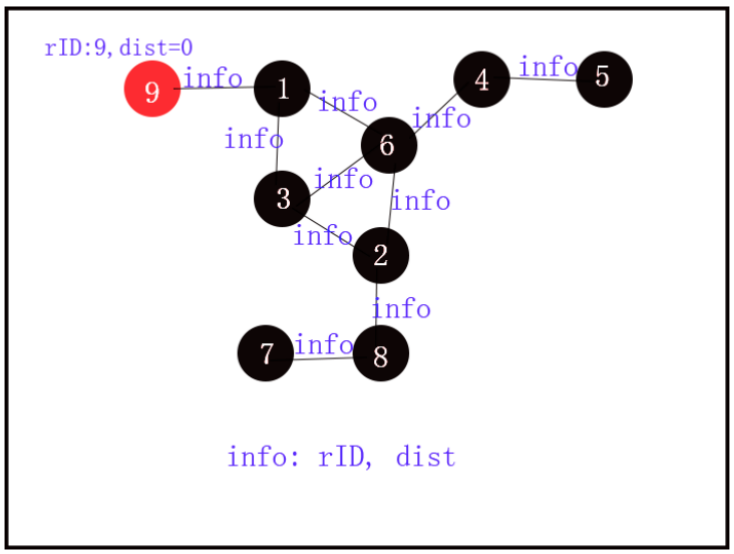


Figure 3.2: Finding Root Sensor 2

received from its neighbor and the rID stored in its own memory, sensor 9 will keep its original rID (9) and distance (dist=0).

As shown in Figure 3.3, sensor 1 will receive messages from sensors 9, sensor 3, and sensor 6. Sensor 6 sends information (rID:6,dist=0) to sensor 1 and sensor 3 sends information (rID:3,dist=0) to sensor 1. Also sensors 9 sends information (rID:9,dist=0) to sensor 1. So three IDs have been sent to sensor 1. After comparing the rIDs, it will update information (rID=9, dist: from sensor1 to sensor9) as the latest information stored in its memory. As we can tell that 9 is the largest ID in the graph, sensor 1 will keep rID=9 until the program is finished.

Similarly, sensor 3 receives three IDs of information from its three neighbors: sensor 1, sensor 2, and sensor 6. After comparison, sensor 6 will be selected as sensor 3's root sensor and the distance between them is 1. Due to update of sensor 1, sensor 1 will send information (rID:9,dist=1) to sensor 3. Sensor 3 receives information from sensor 1 and updates the root sensor's information stored in its memory, which is information (rID:9,dist=2). By the same principle, sensor 2 will select sensor 9 as its current root sensor with the distance set equal to 3 because the distance between sensor 9 and sensor

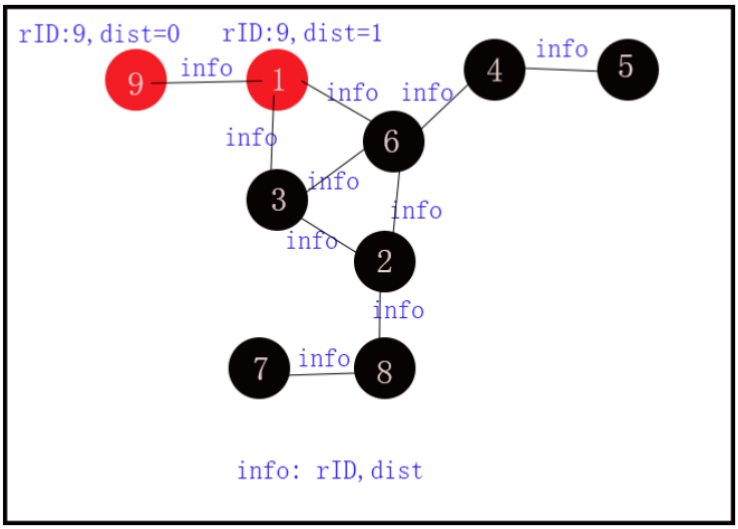


Figure 3.3: Finding Root Sensor 3

3 is 2 and the distance between sensor 3 and sensor 2 is 1. In Figure 3.4, the graph shows sensor 3 and sensor 2 update their root ID information after a few iterations.

Eventually, after a set of iterations or comparisons, the sensors will not update their root sensor information, which means all nodes know the root sensor (the sensor with largest ID number) and the distances between them and the root node. Figure 3.5 shows the final status of finding root sensor.

3.1.2 Breadth First Search (BFS) for the Spanning Tree

After selecting the root sensor, we use the Breadth First Search algorithm for constructing a spanning tree. BFS is an extremely useful searching technique. It differs from the depth-first search in that it uses a queue to perform for search, so the order in which the nodes are visited is quite different. It also has the a very useful property that the first time a node is visited uses the shortest path between that node and the root node. Starting from the root node, it firstly searches adjacent nodes and puts its adjacent nodes into a First-In-First-Out(FIFO) queue. Then it compares distances between it and its neighbors. For all of its neighbors, we set the order of neighbors in terms of the sensor ID. After comparing

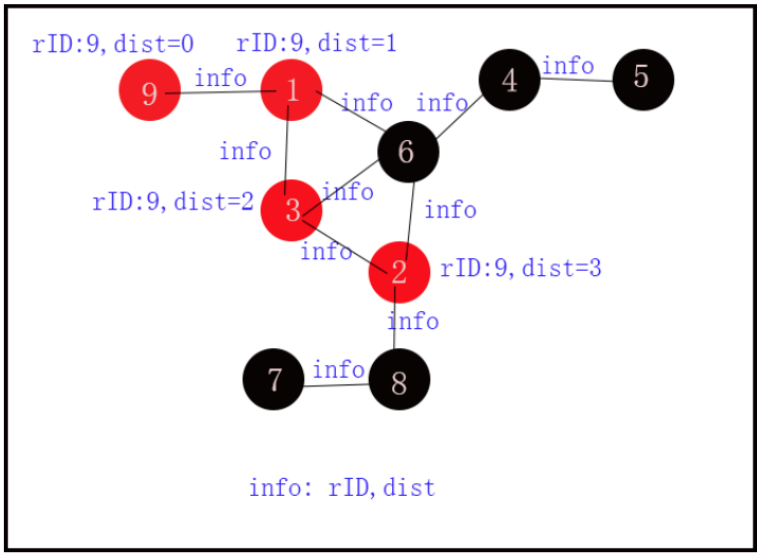


Figure 3.4: Finding Root Sensor 4

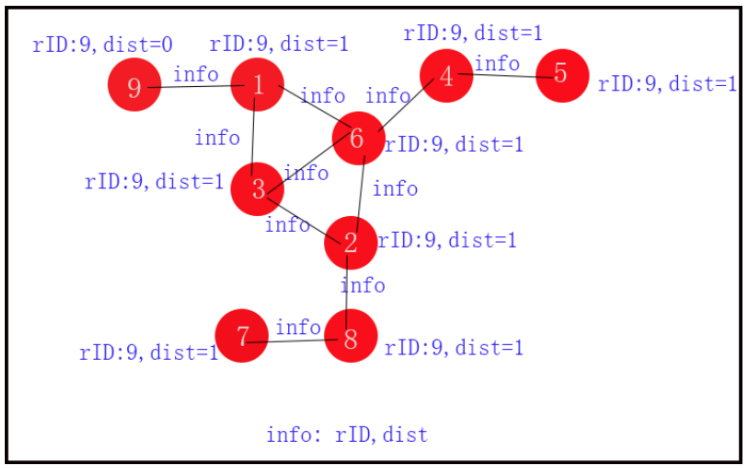


Figure 3.5: Determine the Root Sensor

the sensors' IDs, starting from the node whose ID is the largest number, the node will be found and deleted from the queue. So when all the neighbors have been visited, they are deleted from the queue and marked as visited. Therefore, if other nodes are searching for the neighbors, they will not enqueue the neighbors that have been marked as visited. It is very important that each node should not be visited twice or more. Similarly, each sub-node looks for its adjacent sub-nodes. After a set of iterations, as soon as all the nodes are visited, the spanning tree of the network connectivity graph is established. The following code shows the pseudocode of BFS. G is the graph of network connectivity and v is the node that is searching the sub-nodes.

Algorithm 2 Breadth First Search

```
1: procedure BFS(G,v) is
2: let Q be a queue
3: Q.push(v)
4: lable v as discovered
5: while Q is not empty do
6:     v = Q.pop()
7:     for all edges from v to w in G.adjacentEdges(v)
8:     if w is not labled as discovered then
9:         Q.push(w)
10:        lable w as discovered
11:    end if
12: end while
```

In Figure 3.6, beginning from node 9, it finds its single neighbor node 1. After that, node 1 finds two neighbors, node 3 and node 6. Assuming that the distance between node 6 and node 1 is same as the distance between node 3 and node 1, node 6 will search its neighbors first. Node 2 and node 4 are found and node 2 is selected to search its neighbors first because of its smaller distance. Even though node 3 is its neighbor, node 3 has been visited. Therefore for node 2, there is only one available neighbor. So node 8 is visited. And then node 8 searches the only available node in the figure which is node 7. Repeatedly,

node 4 and node 5 are found consecutively by node 6 and node 5, respectively. Since node 2 and node 6 have been visited already, node 3 has no sub-nodes.

Since a Breadth First Search (BFS) prefers the shortest path, the tie is broken by messages with smaller distance. So the spanning tree is shown below:

3.2 The Gravity Center

To make sure that the sensors move less distance to form a stable network, we have to find the gravity center of the network. There are two processes in this step: the first is convergence process and the second is selection. The convergence process is used to gather the information (each sensor's sub-nodes number). It is also a base for the second process because it makes it easier to calculate the largest sub-branch. As for the selection process, through comparing the sensors' largest sub-branches, we can get the gravity center whose largest sub-branch is smallest among the sensors.

3.2.1 The Convergence Process

As soon as the spanning tree is constructed, every node of the tree stores its parent node information in its memory and each leaf node of the spanning tree sends information including its ID and its number of sub-nodes (including itself) to its parent node. After all leaf nodes of the tree finish sending messages to their parent nodes, similarly, the other nodes except the root node send information to their parent nodes. This process repeats until the root sensor receives information on all its branch nodes, which means all sensors in the spanning tree network have sent messages including their information to their parent sensors. In Figure 3.8, starting from the leaves of the spanning tree, each node sends a message to its parent nodes. The first number represents the number of their sub-nodes (including themselves) and the second number represents the sensorID. The process of convergence is presented by the pseudo of Algorithm 3.

After the convergence process, every node of the spanning tree knows its number of sub-nodes. Each node has many branches and the number of branches is decided by the

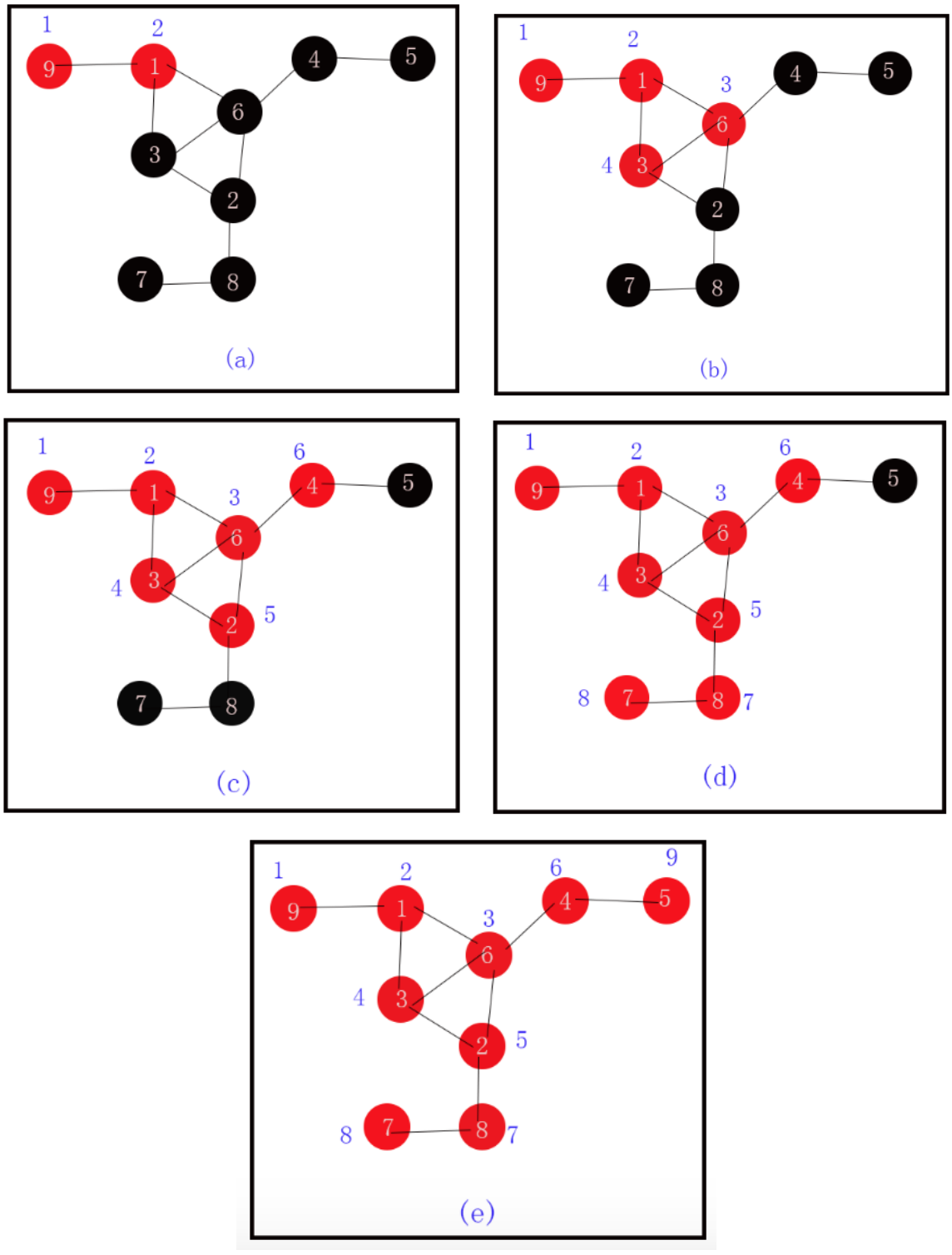


Figure 3.6: Breadth First Search (BFS)

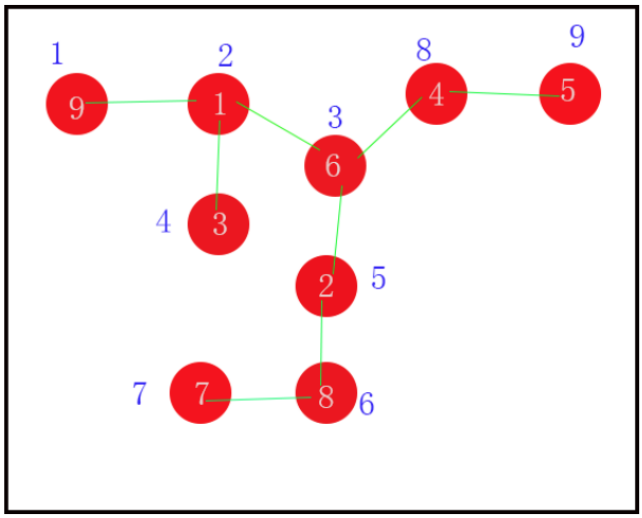


Figure 3.7: The Spanning Tree

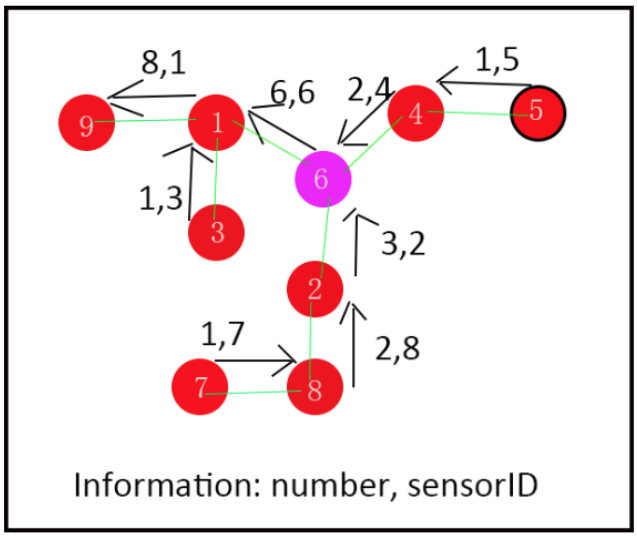


Figure 3.8: Convergence Process

Algorithm 3 Convergence Process

```
1: for all the sensors in network (G) do
2:     if the node(v) is leaf node  then
3:         send message to parent node
4:         label v as sent node
5:     end if
6: end for
7: while true do
8:     for all the sensors in network do
9:         if v is not leaf node & not root node then
10:            send message to parent node
11:            label v as sent node
12:        end if
13:    end for
14:    if number of sent sensor is equal to (G-1) then
15:        stop running the convergence process
16:    end if
17: end while
```

number of its direct sub-nodes. As all nodes in network already know their number of sub-nodes, so they also know the number of nodes in each branch. This is a vital point for the next step: selecting the Gravity Center.

3.2.2 Selecting the Gravity Center

After the spanning tree graph has been constructed, starting from all leaves of the tree, each node sends the number of its branches including itself to its parent. After the convergence process is completed, all of the nodes will know the number of sensors in their own branches. Then we can select the centroid of the spanning tree of the network graph as the gravity center. The reason why we select a gravity center is that each of the other sensors do not need to travel a long distance to surround the selected gravity center. As in the GRG algorithm, all sensors are surround the Point of Interest (POI), so sensors in our algorithm will also move around the gravity center, which is equivalent to the POI. Due to the shortened distance of travel, sensors can conserve their energy for longer lifelimes and also the company can save much money on sensor batteries.

According to the nature of the spanning tree, it has two types: Single-centroid and Bicentroid. When a spanning tree only has one centroid, it is a Single-centroid spanning. A spanning tree is said to be Bicentroids if it has two centroids. There are two algorithms for selecting the gravity center (or centroid) of a spanning tree. One is removing the nodes of degree one repeatedly, and the other one is counting the number of nodes in the sub-tree of each node. For the first method, specifically, there are two steps. The first is removing all the vertices of degree 1, together with their incident edges, and the second step is repeating the process until we obtain either a single vertex (the center) or two vertices joined by an edge (the bicentroid). Figure 3.8 shows how the algorithm runs specifically. First in (a), as node a, node b, node d and node g's degrees are one, so we remove them from the graph and then we get figure (b). In Figure 3.8 (b), there are two nodes of degree one, which are node c and node f. After deleting them, we get the gravity center of spanning tree, which is node e shown in Figure 3.8 (c). These three graphs show the circumstance that there is a single centroid spanning tree of network connectivity with the first method for selecting

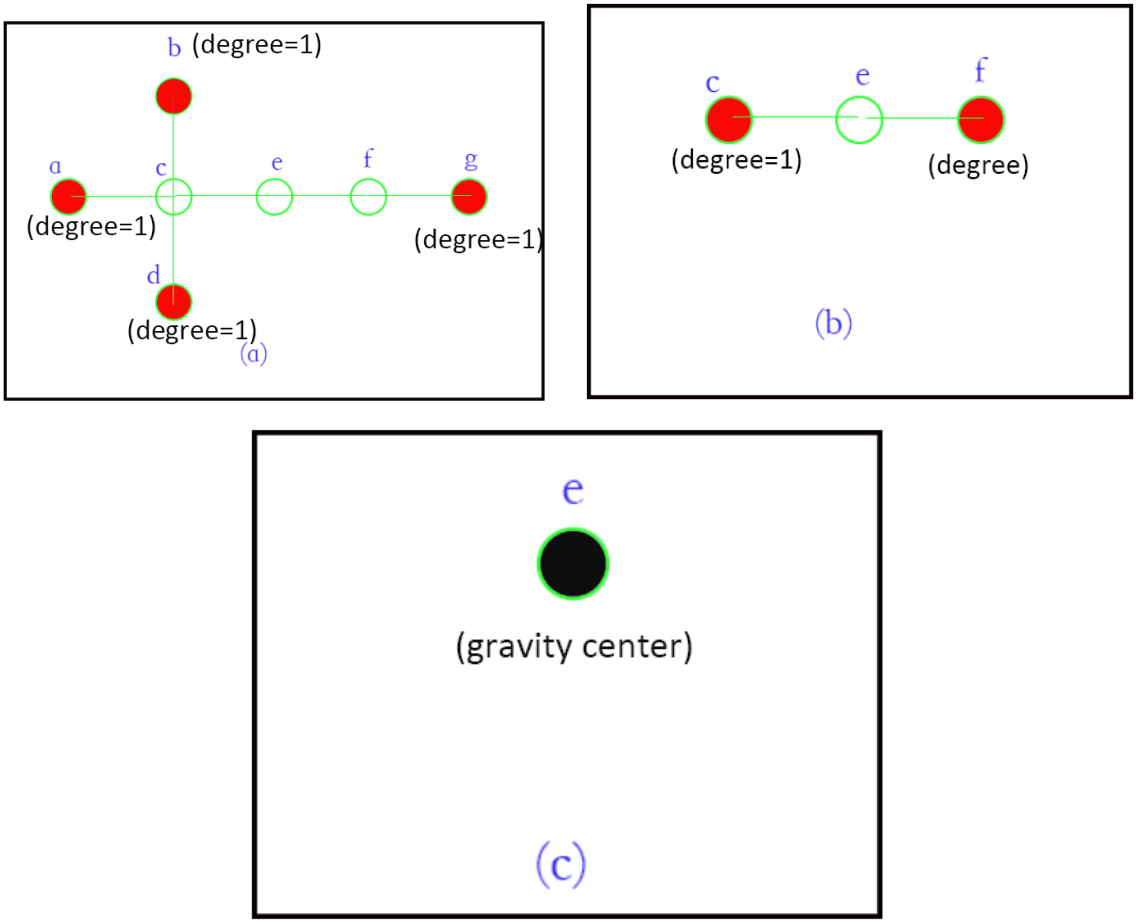


Figure 3.9: Selecting Gravity Center 1

the gravity center.

In our thesis, we use the second algorithm. The Algorithm 4 shows the pseudo code for selecting gravity center process. For each vertex v of degree 2 or more, count the number of vertices in each of the subtrees emanating from v, and let n_V be the maximum of these numbers. The vertex whose n_V is the smallest among the vertices in the region is the gravity center or centroid of the spanning tree. If the tree has n vertices it can be shown that either there is just one centroid v for which $n_V \leq \frac{n-1}{2}$ (the centroid or centroid tree) or there are two adjacent centroids v and w for which $n_V = n_W = \frac{n}{2}$ (the bicentroid or bicentroid tree). It is easy to see that every tree is either centroidal or bicentroidal, but not both. In our previous spanning tree graph, for example, the root node is node 9 and it has

Algorithm 4 Selecting Gravity Center

```
1: let A be the Arraylist of nodes and g be the first node in the list
2: while ture do
3:     check each node's largest branch
4:     if node v's largest branch is smaller than g's branch then
5:         let v=g
6:         each node is marked as checked
7:     end if
8:     if all nodes have be checked then
9:     end if
10: end while
11: select g as gravity center
```

9 sub-nodes including itself, which means $n_9 = 9$. Similarly, node 7, node 3, and node 5 have the same number of sub-nodes and they are in the same situation. Additionally, it is obvious that node 1's largest number of nodes in sub-tree (n_1) is equal to 7. Accordingly, $n_6 = 4$, $n_2 = 7$, $n_4 = 8$ and $n_8 = 8$. So according to method 2, the gravity center of this spanning tree is $node_6$ because n_6 is the smallest value among n_V (V is from 1 to 9). Figure 3.10 shows the single gravity center (centroid) of the spanning tree.

However, another circumstance is that there are two centroids in the spanning tree. In other words, there are two nodes have the same number of sub-nodes. In this case, we will select the sensor with the larger ID as the gravity center, because in the GRB algorithm, there is only one input in the algorithm. Figure 3.11 shows the scenario of a bi-centroid spanning tree. In the graph, n_C and n_d are same and also they are the smallest numbers. Therefore there are two centroids in the graph. Under this scenario, we will select node C as the gravity center.

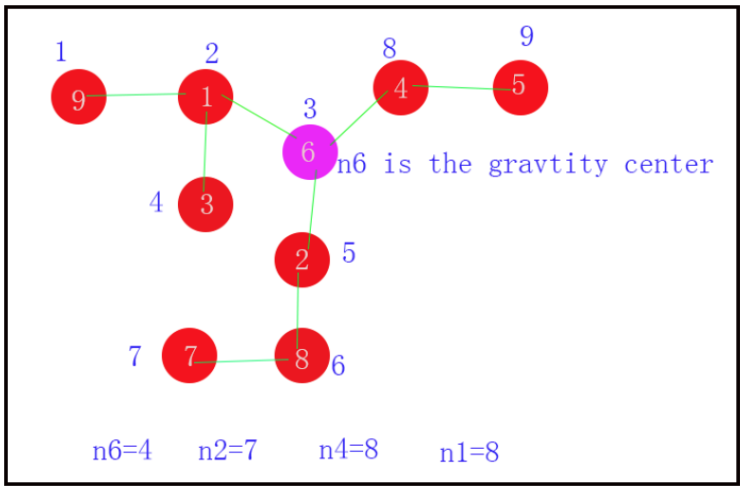


Figure 3.10: Selecting Gravity Center 2

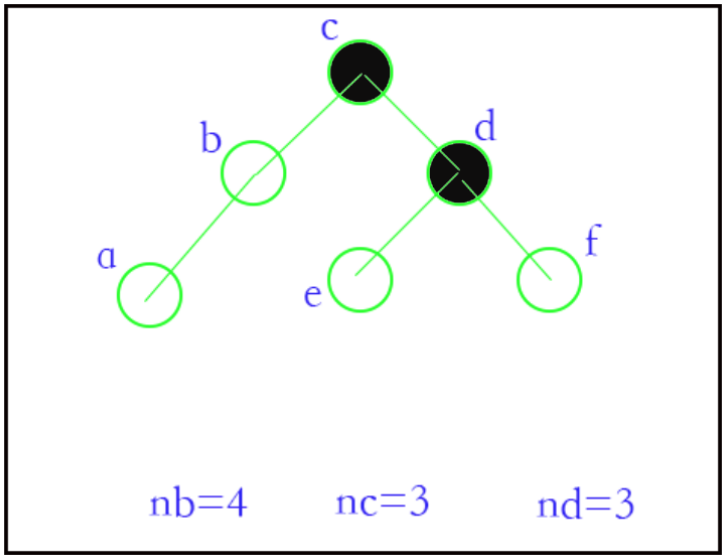


Figure 3.11: Selecting Gravity Center 3

3.3 The Greedy-Rotation-Back (GRB) Method

Once the construction of the spanning tree is completed and the gravity center is selected, the location of the gravity center is broadcast to all the sensor nodes, and the GRB algorithm is run to form the focused coverage. As the region of interest is designed to use a Triangle Tessellation (TT) layout, the sensors will move to the vertices first. Given a common orientation North, and edge length l_e, sensors are able to compute the vertices of the TT layout containing focused coverage. If there is an edge between the two vertices, we say the two vertices are neighboring. As each node knows its own location, it can determine if it is located at a vertex. Therefore, we transfer the sensing coverage issue to occupations of vertices of the TT layout.

The Greedy-Rotation-Back (GRB) algorithm is composed of a set of simple hop selection rules and it is resilient to node failure . Based on these rules, the sensors can make decisions by themselves and, through making a set of decisions, they move toward the gravity center asynchronously. All of the sensors stop moving as soon as there is no available next vertex. When there is a static node located on the vertex or the sensor is moving to the vertex, we define that the vertex is occupied. To introduce the conception of Greedy-Rotation-Greedy, we assume that the sensors are located at different vertices of TT layout (Figure 3.11) initially. As in the practice scenario, the sensors are dropped randomly, so we introduce the Back process for promoting the algorithm. Therefore, in the first part of algorithm, we use the GRG rules [23] and later we will introduce the Back process.

3.3.1 Greedy-Rotation-Greedy (GRG)

The Greedy-Rotation-Greedy algorithm [23] is extended from the Greedy-Advance algorithm [22] and it adds a Rotation movement based on GA algorithm. For the GRG algorithm, there are two more methods for solving collision problems. For the GA movement, the sensors can move greedily to the gravity center and, finally, the sensors surround the gravity center. For Rotation movement, the sensor moves along its current hexagon to find

an available next hop. The position of sensor is represented in a fixed format $< i, j, k >$. i represents the Hexagon, j for Sector and k for Sequence. Specifically, there are 6i vertices with equal TT distance i to gravity center (P) in the TT layout. So the vertices form a distance-i hexagon which is represented by H_i. It is easy to calculate the total number of vertices enclosed by H_i:

$$n(i) = \sum_{q=1}^{i} 6q + 1 = 3i(i+1) + 1$$

There are two types of verteices in the GRG algorithm. The first one is called the corner vertex and the second one is called the edge vertex. As the ROI is designed into a TT layout, so the total area is divided into six sectors. "Sector 0" denoted by S_0, faces to the south and the other sectors proceed from S_1 to S_5 after S_0 in counterclockwise direction. As we previously introduced, corner vertices form the six rays: R_0,...,R_5 in counterclockwise direction. We define that S_p ($0 \leq p \leq 6$) is the area between rays R_p and $R_{(p+1)\%6}$ and "%" represents the modulus operation. For S_q, its counterclockwise next sector and clockwise next sector are $S_{(q+1)\%6}$ and $S_{(q+5)\%6}$ respecticely. The graph of the TT layout is shown in Figure 3.12.

It is apparent that there are two methods for representing the position of the vertex. For the first method, we assign each vertex located on H_i an in-hexagon index called l. So for all of vertices on H_i, their numbers are from 0 to $6i - 1$, which are represented by $< i, 0 >, < i, 1 >, ..., < i, 6i - 1 >$. Considering the second method, we assigned each vertex located on H_i an in-sector index that is k. The format for recording the position of the vertex is $< i, q, k >$, where i is the rank of the hexagon and q represents the number of sector. Additionally, k is the in-sector index, which is from 0 to $i - 1$.

As there are many sensors in the ROI and we cannot predertermine the results of deployment, we need to establish a set of rules to regulate the sensors' movement. In the GRG algorithm, there are seven rules for sensors.

Rule 1(Priority Rule): If there are two nodes moving greedily to $< i, q, k >$ from $< i + 1, q, k >$ and $< i + 1, q, k + 1 >$ (or $< i + 1, (q + 5)\%6, k >$ if k=0), the sensor on

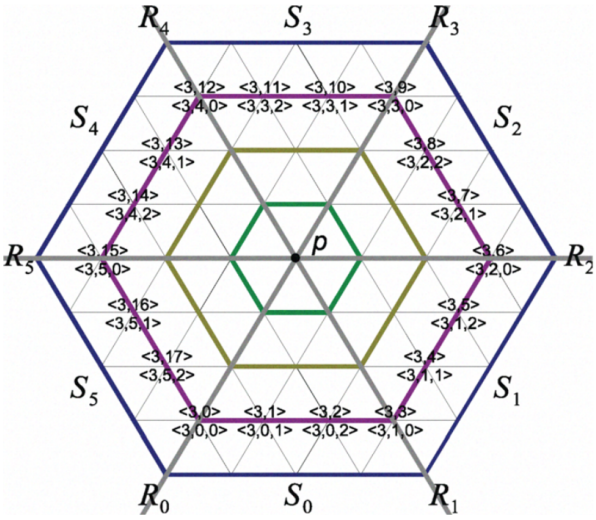


Figure 3.12: Triangle Tessellation (TT) Layout

vertex $< i+1, q, k+1 >$ (or $< i+1, q, k >$ respectively) has higher priority to move to vertex $< i, q, k >$.

In Figure 3.13, node 4(located on $< 3, 3, 1 >$) and node 5(located on $< 3, 3, 2 >$) move greedily to vertex $< 2, 3, 1 >$ at the same time. Due to the Priority Rule, node 5 has higher priorty to move to vertex $< 2, 3, 1 >$ and therefore node 4 cannot move to vertex $< 2, 3, 1 >$ in this case. However, there exists a special circumstance when two nodes move to $< i, q, 0 >$ from $< i+1, (q+5)\%6, i >$ and $< i+1, q, 1 >$. As they are not adjacent to each other, they are not aware of each other and have no idea of one another's information. So in this case a collision may happen on vertex $< i, q, 0 >$. To avoid this situation, Forbiddance Rule is introduced.

Rule 2 (Forbiddance Rule): A sensor located at $< i+1, q, 1 >$ does not choose vertex $< i, q, 0 >$ as its next greedily moving position. In Figure 3.14, the graph shows the Forbiddance Rule. Node 9 and node 8 are not neighbors, so node 9 does not know information about node 8, and vice versa. The Forbiddance Rule is applied in this situation and node 8 cannot move to vertex $< 2, 0, 0 >$. Therefore, node 9 and node 8 move greedily to vertices $< 2, 0, 0 >$ and $< 2, 0, 1 >$, respectively.

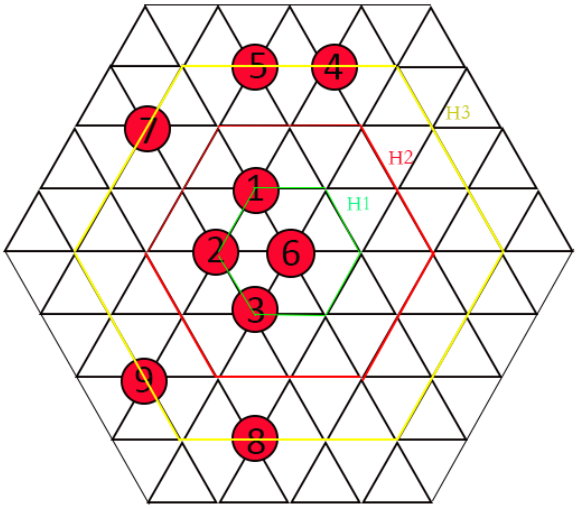


Figure 3.13: Priority Rule

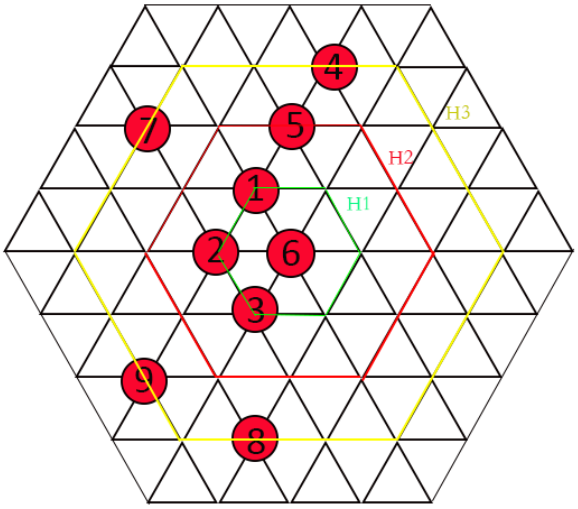


Figure 3.14: Forbiddance Rule.

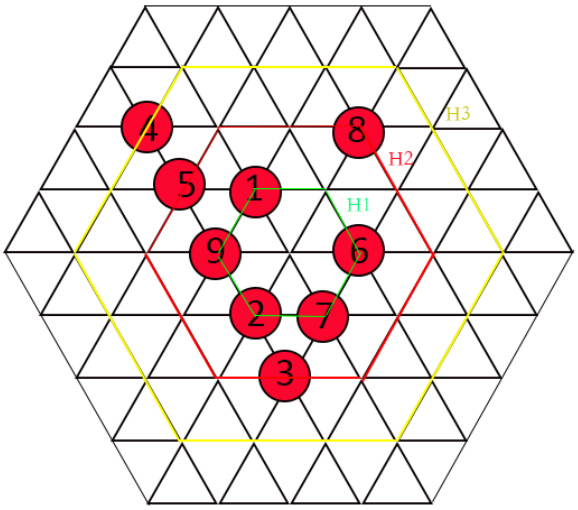


Figure 3.15: Suspension Rule.

Generally, the rotation direction is counterclockwise, which means the node located at vertex $< i, q, k >$ always chooses vertex $< i, q, k+1 >$ (or vertex $< i, (q+1)\%6, 0 >$ if k=i-1). A rotating node stops when it is located at a vertex where the greedy process can resume, or when it reaches its original vertex. As sensors move asynchronously, there exists a special circumstance where a rotating node located at H_i cannot move greedily to an internal hexagon despite some available vertices in H_{i-1} if its internal neighboring node rotates with it. For solving this issue, the Suspension Rule is introduced.

Rule 3(Suspension Rule): A node located at H_{i-1} will rotate if and only if none of its neighbors are rotating on H_i. In Figure 3.15, as there is a sensor (node 1) located on vertex $< 1, 4, 0 >$, so node 5 located on H_2 will execute a rotation process. Before it executes a rotation process, it checks it's neighboring nodes, and since node 4 located on H_3 is rotating, node 5 will stop its rotation process.

Rule 4 (Competition Rule): If a rotating node and a greedily moving node are targeting the same vertex, the latter one keeps its movement and the rotating one will change its deployment plan. However, when a node located at vertex $< i, q, k-1 >$ and a node

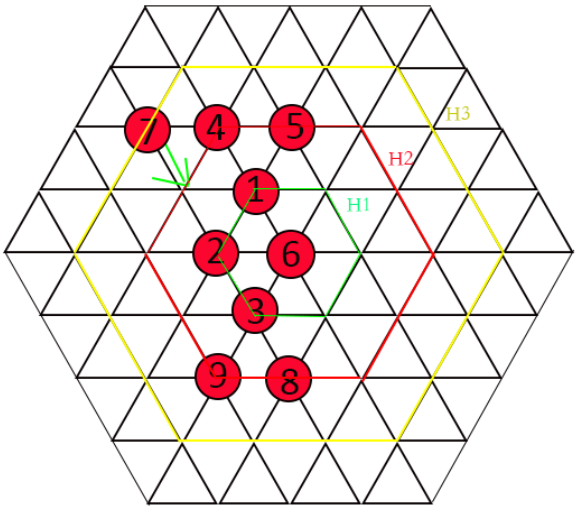


Figure 3.16: Competition Rule

located at vertex $< i+1, q, k+1 >$ move synchronously and both of them are targeting $< i, q, k >$, the former one implements a rotation process and the latter one implements a greedy advance process. Under this scenario, a collision will happen at $< i, q, k >$ because they are not neighboring each other. In Figure 3.16, there is no greedy position for node 4 so node 4 will rotates on H_2 to $< 2, 4, 1 >$. For node 7, it can move greedily to $< 2, 4, 1 >$. In this case, according to the Competition Rule, node 4 will stay at its current postion and node 7 will greedily advance to vertex $< 2, 4, 1 >$.

To avoid this situation, the Edge Rule and the Corner Rule are introduced.

Rule 5 (Edge Rule): A sensor located on an edge vertex $< i, q, k >$ only greedily moves to $< i-1, q, k >$ (or $< i-1, (q+1)\%6, 0 >$ if k=i-1). Because of the Edge Rule, nodes from $< i, q, k-1 >$ and $< i+1, q, k+1 >$ can avoid collision on $< i, q, k >$.

Rule 6 (Corner Rule): A node located at a corner vertex $< i, q, 0 >$ will not move greedily, which means that no greedy advance process in the sensor which is at the corner vertex.

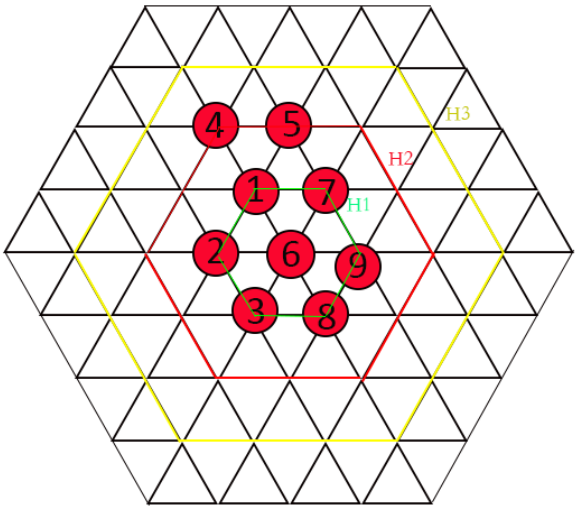


Figure 3.17: Final Status.

According to the previous six rules, the final status of the Algorithm is shown in Figure 3.17.

3.3.2 The Back Process

The GRG algorithm, as previously stated, is designed based on an assumption that sensors initially are dropped at different vertices. However, in practice, sensors are dropped in an ROI randomly and not dropped strictly at vertex positions. Some sensors may be close to each other and some other sensors may have much more space between them. We introduce Back process to fix this problem and also improve the original method for retreat movement.

First we will introduce the Together Rule [23]: sensors located at the edge of the triangle or inside the triangle will move to the triangle vertex occupied by the least number of sensors. If more that one such vertex exists, the sensor will select the closest one. It will select a vertex to move to in case of tie.In Figure 3.18, as vertices around node 1 are occupied by the same number of sensors and the distances between vertices and node 1 are

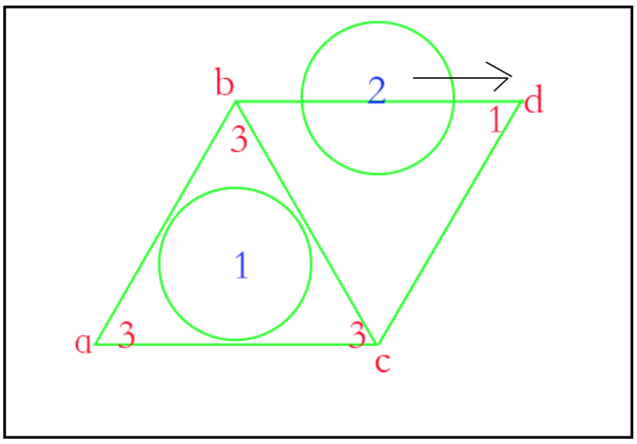


Figure 3.18: Together Rule.

same, node 1 will select a vertex to move to randomly. On the other hand, node 2, located on the edge of triangle, will obviously select vertex d because it is occupied by only one sensor. This rule can cause many collisions on a vertex. We give an order to the sensors that are located at same vertex in terms of IDs. The sensor with higher ID will decide its next deployment first.

As many collisions occur on verteices, to maximize the sensing coverage in ROI, we introduce the Back Rule. As previously introduced, sensors located at same vertex will be ranked and the sensors with highest rank will make its next deployment decision first. If the kth sensor decides to stay at its current position, the sensors that have lower rank than it will decide to implement the Back Rule. Back Rule [23]: When a node located at a vertex on H_i implements the back process, it will move to its neighboring vertex on H_{i+1} that is occupied by least number of sensors. It will randomly select a vertex in case of a tie. In Figure 3.19, there are 4 sensors located at vertex b, which are sensor 2, sensor 10, sensor 12 and sensor 14. Sensor 14 has the largest ID=14, so it has the highest rank and can make a decision first. It decides to rotate to vertex d. After sensor 14 makes its own deployment decision, sensor 12 decides to stay at vertex b. Therefore, sensor 2 and sensor 10 will decide to go back to hexagon H2. As sensor 10 has higher rank, it determines to move to vertex a because a is occupied by 2 sensors. After sensor 10 is deployed, vertex a and vertex c have the same number of sensors. So when sensor 2 implements the back

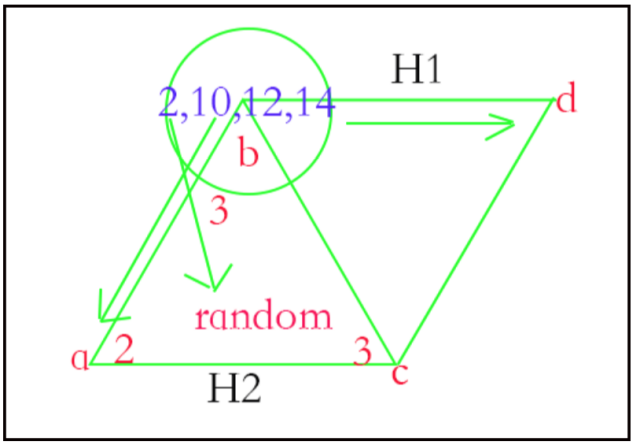


Figure 3.19: Back Rule.

process, it will randomly select vertex to move to for deployment.

When the number of sensors is large in the layout, collisions between sensors cause more distance for sensors to travel. For example, when many sensors are located at one position, they have to wait on sensors who have higher priority (higher sensorsID) than them. As a result, the convergence time will increase. On the other hand, if their greedy move aim position and rotation aim position are occupied by other sensors, they will move back to an outside hexagon, which results in more distance when the GRG process begins. So we introduced Check Rule.

Check Rule: After the Together Rule and before the Back Rule, sensors located at same vertex will check if there is an available potential vertex on their hexagon for rotating or greedy move aim position. Checking the greedy move aim position is very easy to implement by checking the neighbouring sensors' information. To check the potential rotation movement or greedy advance movement, sensor can get information about its neighbouring nodes destination. If it will move immediately, the sensor will stay and waiting for the potential vertex to become available.

As shown in Figure 3.20, sensor 3 and sensor 8 are on same vertex. It is apparent that there is no available vertex for sensor 3 and sensor 8 to greedy move. Meanwhile, as we illustrated previously, sensor 8 has higher priority than sensor 3 and will decide the

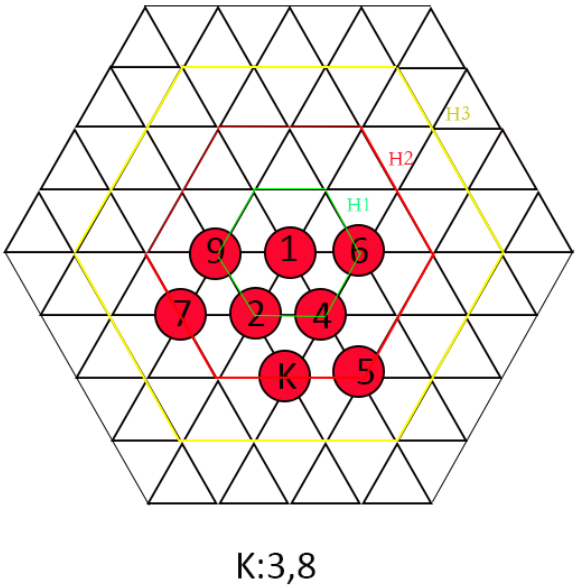


Figure 3.20: Check Rule.

next plan first. As there is no rotation and greedy move positions for sensor 8 to make, it decides to stay at the current vertex. According to the Check Rule, before sensor 3 moves outside the hexagon (H3), it checks the information from sensor 5 and knows that there is a potential position for sensor 8. As a result it also decides to stay at its current position. This can let sensor 3 move less distance and avoid more collisions. If there are sensors located on back positions, when it moves back this will trigger new collisions.

Chapter 4

Simulation Results

In this chapter, we evaluate the performance of the STGRB(Spanning Tree-based Greedy-Rotation-Back) Algorithm. Our simulation is written in Java with Jbotsim library relied upon JDK 1.7, which is a very convenient tool for us to do simulate sensor deployment. With this library, we can evaluate the protocol easily because there are many methods and functions for us to use. We do not have to define many methonds and functions to finish our program, and therefore we can also run our program in a visible environment. As shown in our program, we can see sensors move in the region. For the sensor deployment, we compare the performance of the STGRB algorithm with the existing the GRG algorithm. As we introduced in the previous chapters, in the STGRB algorithm, we used the Breadth First Search(BFS) method for constructing the spanning-tree of network and through removing the sensors with higher degree, we get the gravity center of network. According to the position of gravity center, we run the GRB process. the GRB process is derived from the GRG algorithm, and the Back process is changed based on the GRG algorithm for better performance. Both of the two algorithms are using the Triangle Tessellation layout. That's why we compare these two algorithms. First step, we will introduce the metrics of evaluation. After the metrics are introduced, we will explain the network configuration and introduce the simulation steps. Finally, we will get the figures of simulation and analyze the results. Through comparing these two algorithms, we can get the straight results and get the advantages and disadvantages of the algorithm.

4.1 Simulation Metrics

There are three metrics in our simulations, which are convergence time, move distance, and message cost. Through these three metrics, we can evaluate the protocol comprehensively.

• Convergence time (T): The convergence time is defined as the time that the sensor self-deployment needs for constructing a stable network. For example, when the number of the sensors n is not equal to the $V(k)$, once all the vertices of $V(k-1)$ are occupied by sensors, we think that the region of interest is covered by all the sensors.

• Move distance (MD): Move distance is used to estimate the total distance of a sensor during the whole self-deployment. This metric is used to assess the energy consumption of the sensors. In our simulation, the sensors' moving speed is constant and the less distance the sensors move, the less energy they consume.

• Message Cost (MC): Message Cost is used to counter the number of messages that sensors pass to each other during the sensor self-deployment. As sensors move asynchronously, sensors have to send messages to each other for communcating so that sensors can avoid collision at the same vertice. As a result, MC will increase correspondingly.

• Crash of sensors (CS): When sensors are present at some place (their coordinates are the same), we consider collisions to happen between them. For example, as we drop sensors randomly at first, some sensors are placed at the positions whoses coordinates are similar to each other. According to the Together Rule which we introduced previously, sensors whose coordinates are similar may move to the same vertice. As a result, collisions just emerge at this point.

4.1.1 Network Configuration

We implement out protocol in a specific region which is a 500m × 500m rectangle. We randomly drop sensors in the 252m × 252m region. We assumed the sensing radius (r_s)and communication radius (r_c) of all sensors to be the same, the $r_s = 10m$ and $r_c = 18m$ (because $10 \times \sqrt{3} \approx 18$), respectively. The length of each edge of virtual TT is l = r_s =

18m. With this length, if all sensors are located at different vertices, the coverage should be maximized automatically. The reason for this is shown in thesis [23]. That is why we always want to generate TT networks in application. Additionally, the size of the network can be replaced by the size of the hexagon, which means the number of the vertices included in the largest hexagon. As we presented in Chapter 3, the number of vertices included in hexagon can be calculated through the formula: $n(i) = \sum_{q=1}^{i} 6q+1 = 3i(i+1)+1$. However, when sensors are required to be placed in some extreme environments, for example, under the sea or in a single forest, some sensors may report location errors. So we just assume our protocol is implemented in a two-dimensional flat region. As when the gravity center may locate at a remote position, and the TT layout cannot be complete in the region, so we do not consider this situation in our simulation.

As we said that the size of network is changed according to the number of hexagons, so the number of vertices changes with the variation of the number of hexagons. $n_(i)$ represents the number of vertices included in the largest hexagon and i represents the number of the largest hexagon. We change the size of the network from $n_(1) = 7$ to $n_(7) = 169$.

Table 4.1: Network Size

Network Size	Formula	Number of Vertices
$n_(1)$	$n(1) = \sum_{q=1}^{1} 6q + 1 = 3 * (1 + 1) + 1$	7
$n_(2)$	$n(2) = \sum_{q=1}^{2} 6q + 1 = 3 * 2 * (2 + 1) + 1$	19
$n_(3)$	$n(i) = \sum_{q=1}^{3} 6q + 1 = 3 * 3 * (3 + 1) + 1$	37
$n_(4)$	$n(i) = \sum_{q=1}^{4} 6q + 1 = 3 * 4 * (4 + 1) + 1$	61
$n_(5)$	$n(i) = \sum_{q=1}^{5} 6q + 1 = 3 * 5 * (5 + 1) + 1$	91
$n_(6)$	$n(i) = \sum_{q=1}^{6} 6q + 1 = 3 * 6 * (6 + 1) + 1$	126
$n_(7)$	$n(i) = \sum_{q=1}^{7} 6q + 1 = 3 * 7 * (7 + 1) + 1$	169

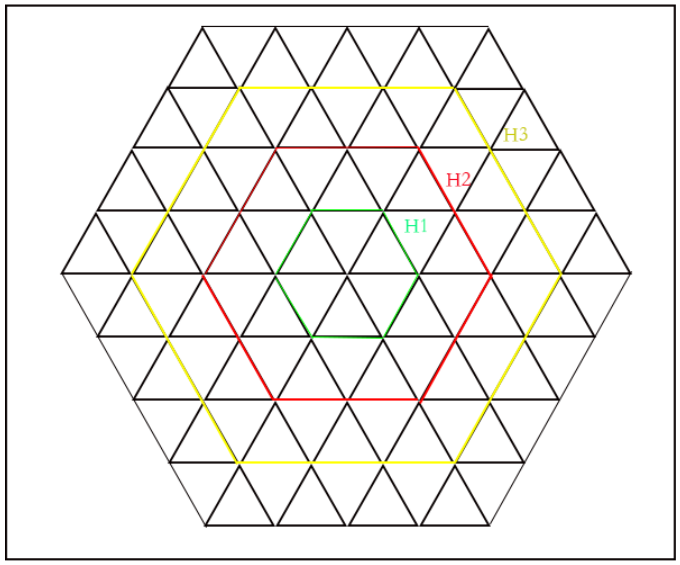


Figure 4.1: Simulation Layout

Our protocol is implemented with some specific parameters, which are shown in the Table 4.2 below. We run our simulation 20 times with different initial positions in one network size. Therefore we can get the average value in each network size scenario.

Table 4.2: Network Configuration

Parameters	Definition	The Values
$n_(i)$	Network Size	7, 19, 37, 61, 91, 126, 169
r_c	Communcation Range	18
r_s	Sensing Range	10
S	Sensors' speed	2m to 3m per time unit
D	Message Delay	1 time unit

The environment is set up like Figure 4.1, where the layout is Triangle Tessellation (TT). Initially, the sensors are dropped on the platform randomly.

In Figure 4.1, H1 is corresponding to the $n_(1) = 7$, H2 is corresponding to the $n_(2) = 19$ and H3 is correspoding to $n_(3) = 37$. We only illustrate three hexagons in the figure and

it is apparent that the vertices can also be calculated by counting. Our size of the region is fixed and the layout in the region is also fixed. The number of the largest hexagon is 7 and the number of vertices should be 169.

4.2 Results and Analysis

In this section, we will illustrate our simulation results, which are shown in figures below, so we can get some direct results from the figures. What we can get from these results is that the GRB algorithm has better performance in terms of Move distance and Crash of sensors.

4.2.1 Convergence Time

For getting the Convergence Time, we calculate the time starting from when all the sensors are dropped on the layout and ending when all the vertices of the inner hexagon are occupied by sensors. Here we define the inner hexagon as the hexagon who is beside the largest hexagon and whose location is closer than largest hexagon's location. As a short period of time is needed for sensors to drop, we start to calculate time after the drop has finished. As we add some methods for deployment, the time needed for deployment in STGRB is longer than in the GRG algorithm. For the finishing time, when the number of sensors n is not equal to $n_(i)$, we stop counting when the inner hexagon is full of sensors, which means every vertice on the hexagon is occupied by one sensor. On the other hand, when the number of sensors n=$n_(i)$, we stop counting when the outmost hexagon is full of sensors. In the GRG algorithm, the POI (point of interest) is located at the center of region, so all sensors (except the sensor already located at the POI) move around the POI. For STGRB algorithm, the POI is not firmly at the center of the region. Every time we run the protocol, the POI may present at a different position. After all sensors finish the GRG process, we let all sensors move until the sensor located at the POI previously arrives at the center of the region. When the process finishes, the timer in our computer ends. It is apparent that the length of Convergence Time is mainly decided by the number of sensors

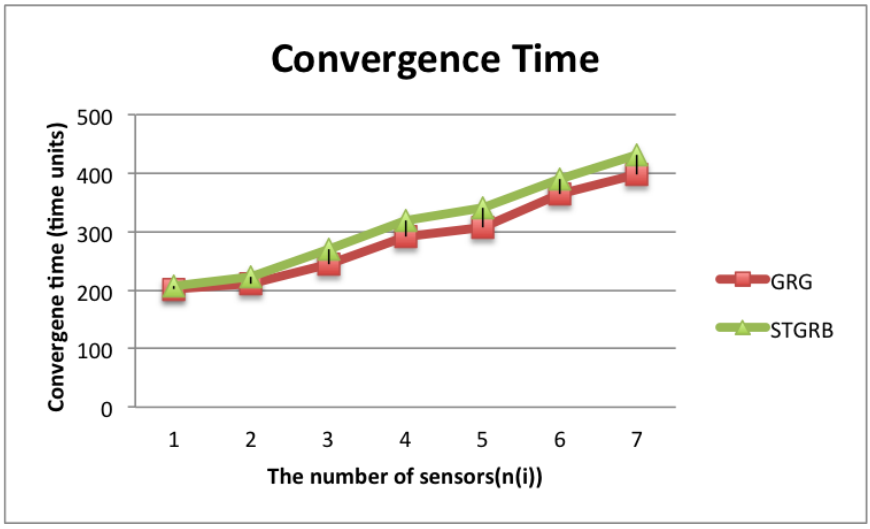


Figure 4.2: Convergence Time

and the location of the POI in the STGRB algorithm. So with the variety in number of sensors, the Convergence Time will change accordingly.

In Figure 4.2, with the increase of the number of sensors, the number of collisions between sensors and their movement time will rise. As a result, the time needed for the self-deployment is longer. From Figure 4.2, we also can see that Convergence Time in the STGRB protocol is longer than Convergence Time in the GRG algorithm. This is because in the STGRB algorithm, we have to construct a spanning tree for the network, then we have to find the gravity center of network based on the spanning tree constructed. These two processes will consume some time during the sensors' self-deployment. In this case, the STGRB algorithm outperforms the GRG algorithm by approximately 30 time units.

4.2.2 Move Distance

In this metrics, we use the number of moves to evaluate the move distance because, in our simulation, the lengths of all the edges of the triangles in the layout are equal to each other. Accordingly, a larger number of moves means longer distance that the sensors have traveled. As, initially, we drop sensors randomly on layout, the sensors may not be located at vertices. So the number of moves per sensors may or may not be an integer. After the

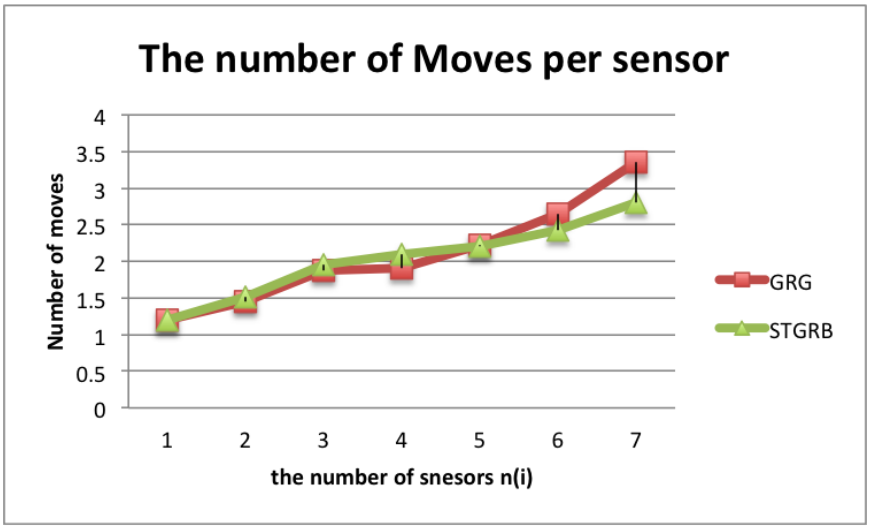


Figure 4.3: The Number of Moves per Sensor

construction of stable network, STGRB nodes move to the same destination as GRG in the region. As in the Back process, sensors that are not located at vertices should move to vertices around them first. That is why the number of moves of sensors is not always an integer.

As shown in Figure 4.3, the number of moves per sensor is increasing with the number of sensors. Like we said before, when the number of sensors is not large, like $n_(1)$ and $n_(2)$, the difference in number of moves between GRG and STGRB is not large. That is because when n is not large, the number of collisions is small. However, with more sensors, the STGRB algorithm performs better than the GRG algorithm. Even though the number of moves of STGRB is larger than GRG's when i is equal to 3 or 4, it is more apparent that when number of sensors is larger than $n_(5)$ the number of moves per node in the STGRB protocol is more smaller than the number of sensors in GRG algorithm. We construct the spanning tree of the network and find the gravity center before the sensors move so that all the sensors do not have to move a long distance to form a stable network. As a result, when sensors move to center area of the region, the total number of moves should be smaller than the number in the GRG algorithm. As we assumed before, the number of moves represents the sensors' moving distance and, accordingly, represents the energy comsumption of the sensors. Finally, from the data, we can easily judge that STGRB

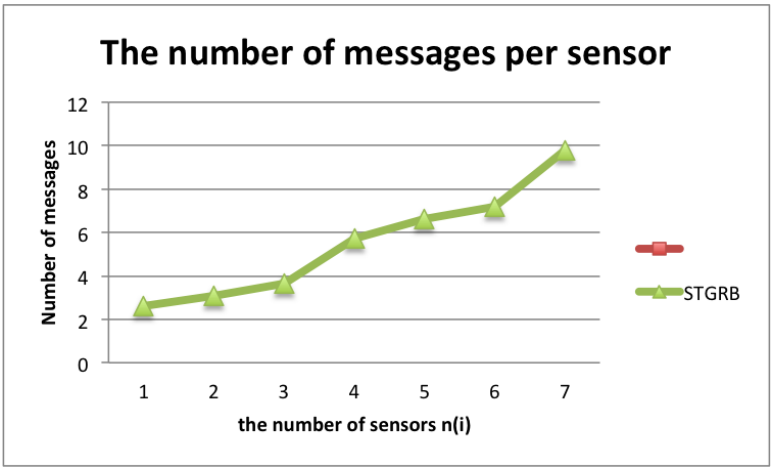


Figure 4.4: The Number of Messages per sensor

algorithm performs better than the GRG algorithm in terms of moving distance.

4.2.3 Message Cost

As we introduced in chapter 3, for the STGRB algorithm, when sensors construct the spanning tree and find the gravity center, the sensors have to propagate messages to each other to get information from their neighbours. For tree construction, the message, including sensor ID, is sent from sensor to sensors to make sure the sequence of sensors in the network is recorded. As a result, the root sensor is decided and the BFS method is used to construct the spanning tree. In the gravity center process, the sensors send information including their number of sub-nodes to find the node whose largest sub-branch is smallest in the tree. Consequently, the gavity center is decided and other sensors move around it. However in the GRG algorithm, message cost is not considered in simulation.

In Figure 4.4, the green line is the message cost of the STGRB protocol. From the figure, we can tell that the message cost of STGRB algorithm rises when the number of sensors increases. As GRG/CV algorithm does not consider the message cost in simulation, here we only have the graph of STGRB algorithm. Generally, the message cost of the STGRB is high. As we can see from the graph, the message cost happens during the spanning tree process and selecting gravity center process. Also as we used flooding broadcast method

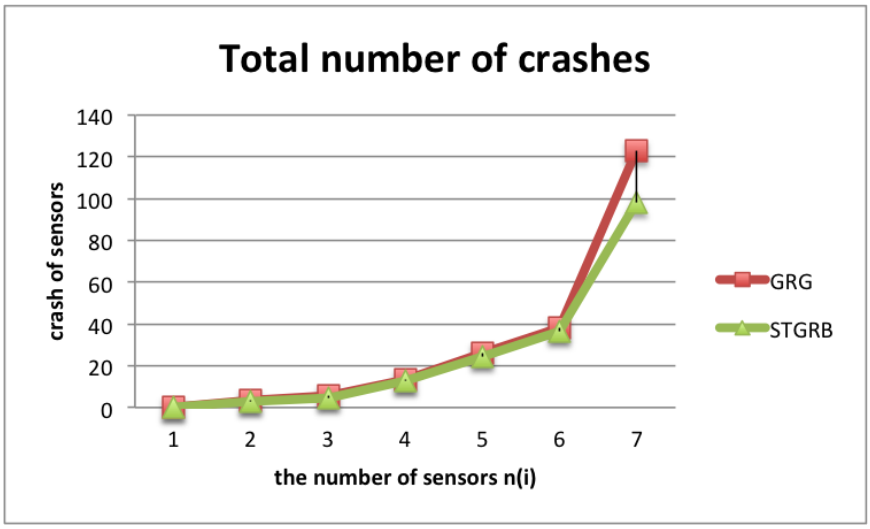


Figure 4.5: Total Number of Crashes

for informing the positions of root sensor and gravity center, the message cost is high when the number of nodes is large enough.

4.2.4 Sensor Crashes

In our algorithm, collisions only happen during the Back process of sensor self-deployment. According to the Together Rule in the Back Process, sensors will move to the vertex of the triangle with the fewest sensors. Consequently, crashes between sensors will happen during the movements. Additionally, the Back Rule of the STGRB algorithm will make sensors move to vertices located at outside hexagons. Accordingly, more crashes between sensors are produced in this scenario.

Figure 4.5 depicts the number of crashes between sensors for the STGRB and GRG protocols. As shown in figure, the green line(STGRB) is always under the red line(GRG), which means the STGRB performs better than GRG algorithm in terms of the number of crashes between sensors. Even though the difference is not clear before the number of sensors n is less than $n_(2)$, with n rising, the STGRB has better and better performance. This is because we add Check rule in Back process. Before a sensor executes the Back Rule, according to the Check Rule, sensors will check the status of its neighbouring node.

Chapter 5

Conclusion

5.1 Conclusion and Future Work

In this thesis, we resolved the self-deployment of robotic sensors without an input. For achieving the aim of implementing a self-deployment without input and saving sensor energy, we developed an automatically forming a stable network with good compactness protocol based on GRG algorithm: STGRB. This protocol uses a spanning tree and a gravity center to reduce sensors' moving distance, and removes the pre-condition of the GRG algorithm. According to Sensor ID, the spanning tree of the network is constructed and each sensor in the network knows its parent sensor. Through the convergence process, every node in the tree knows each other node's number of sub-nodes. By computing the number of sub-nodes, each node in the tree knows its largest branch (the branch with largest number of sub-nodes) and how it compares to other nodes' largest branches. Finally, the node whose largest branch is smallest among the nodes is selected as the gravity center of the network, who is physically close to other nodes(sensors). As the layout in the region of interest is a Tessellation Triangle, which means the sensing coverage is maximized and the overlapped area is minimized in the region, so the sensing coverage of a certain number of sensors is optimized. Intuitively, once the sensor is identified, the location of the sensor is broadcast to all the sensor nodes, and the GRG algorithm is run to form the focused coverage, which is one type of coverage with very good compactness. Therefore our protocol

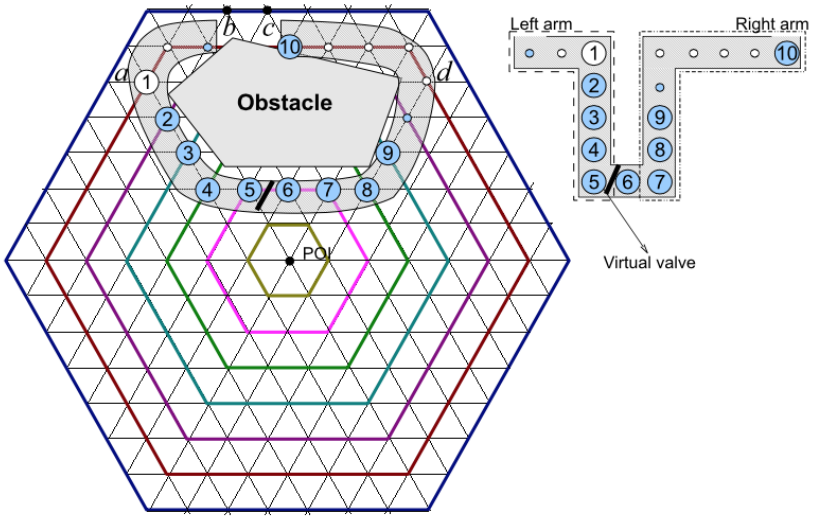


Figure 5.1: U-tube resolution.

is appropriate for some plans that drop a limited number of sensors in an interesting region and form stable sensing coverage automatically. Due to the mobility of sensors, the sensors can form a stable network themselves.

For future work based on our research, there are a few aspects that we can improve. One aspect to improve is to avoid the obstacle in the region of interest. According to [24], a U-tube concept has been introduced to resolve the problem of avoiding obstacles in the ROI. In this way, our protocol can be used in a more complex environment.

The other aspect for improvement is to reduce the number of collisions between nodes, which means developing a new rule for improving the Back process. During the Back process, by useing the new rules sensors can avoid crashes with other sensors and reduce the self-deployment distance.

References

[1] Andrew Howard, Maja J. Mataric, and Gaurav S. Sukhatme. Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem. *In Proceedings of the 6^{th} International Symposium on Distributed Autonomous Robotics Systems (DARS)*, 2002.

[2] S. Yang, M. Li, and J. Wu. Scan-Based Movement-Assisted Sensor Deployment Methods in Wireless Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(8): 1108-1121, 2007.

[3] H. Mousavi, A. Nayyeri, N. Yazdani, and C. Lucas. Energy Conserving Movement-Assisted Deployment of Ad hoc Sensor Networks. IEEE Communications Letters, 10(4): 269-271, 2006.

[4] N. Bartolini, T. Calamoneri, E.G. Fusco, A. Massini, and S. Silvestri. Snap and Spread: A Self-deployment Algorithm for Mobile Sensor Networks. In Proceedings of the 4^{th} IEEE/ACM international Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 451-456, 2008.

[5] S. Chellappan, X.Bai, B. Ma, and D. Xuan. Mobility Limited Flip-Based Sensor Networks Deployment. IEEE Transactions on Parallel and Distributed Systems, 18(2): 199-211, 2007.

[6] R. Ramadan, H. EI-Rewini, and K. Abdelghany. Optimal and Approximate Approaches for Deployment of Heterogeneous Sensing Devices. EURASIP Journal on Wireless Communications and Networking, 2007(1), 2007.

[7] X. Bai, S. Kumary, D. Xuan, Z. Yun and T. H. Lai, Deploying Wireless Sensors to Achieve Both Coverage and Connectivity. In Proc. of ACM MobiHoc, pp. 131-142, 2006.

[8] M. Ma and Y. Yang. Adaptive Triangular Deployment Algorithm for Unattended Mobile Sensor Networks. IEEE Tran. on Computers, 56(7): 946-958, 2007.

[9] H. Zhang and J. C. Hou, Maintaining Sensing Coverage and Connectivity in Large Sensor Networks. Ad Hoc & Sensor Wireless Networks, Vol. 1, pp.89-124, 2005.

[10] W. Ye, J. Heideman and D. Estrin, An energy-efficient MAC protocol for wireless sensor networks. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Vol.1, 2002.

[11] Tijs van Dam and Koen Langendoen, An adaptive energy-efficient MAC protocol for wireless sensor networks. Proceedings of the 1^{st} international conference on Embedded networked sensor systems, pp. 171-180, 2003.

[12] N. Heo and P.K. Varshney. Energy-Efficient Deployment of Intelligent Mobile Sensor Networks. IEEE Transactions on Systems, Man, and CyberNetics – Part A: Systems and Humans, 35(1): 78-92,2005.

[13] F. Aurenhammer and R. Klein. Voronoi Diagrams. http://www.pi6.fernuni-hagen.de/publ/tr198.pdf.

[14] Guiling Wang, Guohong Cao, and Tom La Porta. Movement-Assisted Sensor Deployment. In Proceedings of the 23^{rd} Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Vol.4,pp. 2469-2479, 2004.

[15] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. IEEE Transactions on Robotics and Automation, 20(2): 243-255, 2004.

[16] http://www.math.northwestern.edu/∼mlerma/courses/cs310-05s/notes/dm-spantree.

[17] M. Garetto, M. Gribaudo, C.F. Chiasserini and E. Leonardi. A Distributed Sensor Relocation Scheme for Environmental Conrol. In Proceedings of IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS), pp. 1-10,2007.

[18] S. Poduri, S. Patern, B. Krishnamachari, and G.S. Sukhatme. Using Local Geometry for Tunable Topology Control in Sensor Networks. IEEE Transactions on Mobile Computing, 8(2): 218-230, 2009.

[19] K. Menger. Zur allgemeinen Kurventheorie. Fund. Math., 10: 96-115, 1927.

[20] S. Poduri, G.S. Sukhatme. Constrained Coverage for Mobile Sensor Networks. In Proceedings of IEEE Int'l Conf. on Robotics and Automation (ICRA 2004), 2004.

[21] Pac, M.R., Erkmen, A.M., Erkmen, I. Scalable self-deployment of mobile sensor networks. In Proceedings of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS 2006), 2006.

[22] Kerr, W., Spears, D., Spears, W., Thayer, D. Two formal fluid models for multi-agent sweeping and obstacle avoidance. In Proceedings of AAMAS, 2004.

[23] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. Focused coverage by mobile sensor networks, Mobile Adhoc and Sensor Systems, 2009. MASS '. IEEE 6th International Conference on, pp.466-475, 2009.

[24] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. Localized Sensor Self-deployment with Coverage Guarantee. ACM SIGMOBILE Mobile Computing and Communications Review, 12(2), 2008.

[25] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. Numerische Mathemetik, 1: 269-271, 1959.

[26] A. Howard, M. J. Mataric and G. S. Sukhatme. An Incremental Self-Deployment Algorithm for Mobile Sensor Networks. Autonomous Robots, 13(2): 113-126, 2002.

[27] C. Chang, Y. Chen, and H. Chang, Obstacle-resistant deployment algorithms for wireless sensor networks. IEEE Trans. Veh. Technol, 58(6):2925-2941,2009.

[28] M. Batalin and G. Sukhatme, The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment. Proc. IEEE Int. Conference on Robotics and Automation, pp. 3478-3485, 2005.

[29] X. Li, G. Fletcher, A. Nayak and I. Stojmenovic. Place Sensors for Area Coverage in a Complex Environment by a Team of Robots, ACM Trans. Sensors Networks, 11(1): 3-11, 2014.

[30] H. Mousavi, A. Nayyeri, N. Yazdani, and C. Lucas. Energy Conserving Movement Assisted Deployment of Ad hoc Sensor Networks, IEEE Communications Letters, 10(4):269271, 2006.

[31] A. Cerpa and D. Estrin, ASCENT: Adaptive Self-Configuring Sensor Network Topologies, Proc. IEEE INFOCOM,02, June 2002.

[32] S. Capkun, M. Hamdi and J.P. Hubaux, GPS-Free Positioning in Mobile Ad Hoc Networks, Proc. 34th Hawaii Int' Conf. System Sciences (HICSS ',01), Jan. 2001

[33] Z Liao, J Wang, S Zhang, and X Zhang, A deterministic sensor placement scheme for full coverage and connectivity without boundary effect in wireless sensor networks, Adhoc & Sens. Wireless Networks, Vol. 19, no. 3-4, pp.327-351, 2013.

[34] C. Ozturk, D. Karaboga, and B. Gorkemli, Probabilistic dynamic deployment of wireless sensor networks by artificial bee colony algorithm, Sens., Vol. 11, no. 6, pp. 6056-6065, Jan, 2011.

[35] Y. Wang, A. Barnawi, R. Mello, and I. Stojmenovic, Localized ant colony of robots for redeployment in wireless sensor networks, J. of Multi-Valued Logic & Soft. Comput., Vol. 23, pp.33-51, 2014.

[36] Z. Tu, Q. Wang, and Y. Shen, A distributed self-deployment method for coverage optimization in mobile sensor network, Comm. Networking Conf., pp. 881-886, 2011.

[37] W. Liao, Y. Kao, and R. Wu, Ant colony optimization based sensor deployment protocol for wireless sensor networks, Expert Syst. with App., Vol. 38, no. 6, pp. 6599-6605, 2011.

[38] X. Li, N. Mitton, I. Ryl, and D. Simplot, Localized sensor self-deployment with coverage guarantee in complex environment, Ad-hoc, Mobile Wireless Networks, pp. 1-14, 2009.

[39] C. Chang and J. Sheu, An obstacle-free and power-efficient deployment algorithm for wireless sensor networks. IEEE Trans. Systems, Man and Cybernetics, 39(4): 795-806, 2009.