Sensor Deployment and Coverage Maintenance by a Team of Robots

Qiao Li

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

Acknowledgements

I would like to acknowledge my sincerest gratitude to my supervisor Prof. Dr. Amiya Nayak. Without his the help and support in preparation of this thesis, it would not have been possible to complete this thesis. I am grateful to my previous supervisor Prof. Dr. Ivan Stojmenovic for his enthusiastic and expert guidance during writing this thesis. I would also like to thank my friends and my family in particular for their sustained understanding and support.

Table of Contents

Acknowledgements ..... **ii**

List of Figures ..... **v**

List of Tables ..... **vii**

Abstract ..... **viii**

Chapter 1: Introduction **1**

1.1 Background Information 1

1.2 Problem Statement 2

1.2.1 Sensor Placement 2

1.2.2 Coverage Maintenance by Robots 2

1.3 Existing Solutions 3

1.3.1 Sensor Placement 3

1.3.2 Coverage Maintenance 3

1.4 Motivations and Objectives 4

1.5 Assumptions 4

1.6 Contribution 5

1.7 Organization of the Thesis 5

Chapter 2: Literature Review **6**

2.1 Sensor Self-Deployment 6

2.2 Sensor Placement by Robots 7

2.2.1 Least Recently Visited Approach 8

2.2.2 Obstacle-Resistant Robot Deployment Approach 12

2.2.3 Back-Tracing Deployment Approach 17

2.3 Coverage Maintenance by Robots 24

Chapter 3: Election-Based Deployment Algorithm (EBD) **30**

3.1 Sensor Deployment 30

3.1.1 Strategy Allocation 30

3.1.1.1 Initial Strategy Allocation 30

3.1.1.2 Strategy Reallocation 35

3.1.2 Dead End Situation 44
3.1.3 Gray Sensor Situation 47
3.1.4 Termination of Sensor Deployment Phase 50
3.2 Coverage Maintenance 50
3.2.1 Subarea Combination 50
3.2.2 Choosing Central Points 53
3.2.3 Candidates Selection 57
3.2.4 Worker Election 58
3.2.4.1 Free Candidates 58
3.2.4.2 Busy Candidates 61
3.2.5 Termination of Hole Patching Phase 65
Chapter 4: Simulation Results 66
4.1 Sensor Deployment 66
4.1.1 Simulation Environment 66
4.1.2 Coverage Ratio 68
4.1.3 Robot Moves 69
4.1.4 Workload 69
4.1.5 Robot Messages 70
4.1.6 Sensor Messages 71
4.2 Coverage Maintenance 72
4.2.1 Simulation Environment 72
4.2.2 Robot Moves 73
4.2.3 Robot Messages 74
4.2.4 Sensor Messages 74
4.2.5 Coverage Time 75
Chapter 5: Conclusions and Future Work 76
5.1 Conclusions 76
5.2 Future Work 76
References 78

List of Figures

Figure 2.1 Nodes with Different Directions 9
Figure 2.2 LRV Approach 12
Figure 2.3 Triangle Tessellation 14
Figure 2.4 OFRD Deployment 15
Figure 2.5 Six Basic Movements 15
Figure 2.6 OFRD Approach in Obstacle Scenario 17
Figure 2.7 Incomplete Coverage by ORRD Approach 18
Figure 2.8 "Shortcut" Approach in BTD 21
Figure 2.9 BTD Approach for Single and Multiple Robot Scenario 22
Figure 2.10 Sensing Hole Surrounded by Gray Sensors 23
Figure 2.11 Drawback of BTD Approach 24
Figure 2.12 Message Resending 28
Figure 2.13 Area Partition 28
Figure 2.14 Break of Guardian-Guardee Relationship 30
Figure 3.1 Initial Strategy Allocation 32
Figure 3.2 Drawback of BTD 33
Figure 3.3 Deployment of BTD 35
Figure 3.4 Deployment of EBD 36
Figure 3.5 Crossed Areas 37
Figure 3.6 Rich and Empty Strategy 38
Figure 3.7 Strategy Reallocation 38
Figure 3.8 Reallocation Algorithm 40
Figure 3.9 Picking Algorithm 41
Figure 3.10 First Round of Picking Algorithm 42
Figure 3.11 Second Round of Picking Algorithm 43
Figure 3.12 Fully Covered ROI of Figure 3.5 44
Figure 3.13 Fully Covered ROI of Figure 3.7 45

Figure 3.14 Neighbors of Sensor 0 46
Figure 3.15 Backtracking in EBD 47
Figure 3.16 Backtracking in BTD 48
Figure 3.17 Sensing Hole Encounter 50
Figure 3.18 Combination 53
Figure 3.19 How Combination Benefits Subarea Shapes 54
Figure 3.20 Picking Central Points 57
Figure 3.21 Candidates Selection 59
Figure 3.22 Shortcut-to-Hole Approach 61
Figure 3.23 Columns of Sensing Hole 63
Figure 3.24 Sensing Hole Patching 65
Figure 4.1 Simulation Environment 68
Figure 4.2 Coverage Ratio vs Number of Robots 69
Figure 4.3 Robot Moves vs Number of Robots 70
Figure 4.4 Workload Comparison 70
Figure 4.5 Robot Messages vs Number of Robots 71
Figure 4.6 Sensor Messages vs Number of Robots 72
Figure 4.7 Robot Moves vs Number of Robots 74
Figure 4.8 Robot Messages vs Number of Robots 75
Figure 4.9 Sensor Messages vs Number of Robots 76
Figure 4.10 Coverage Time vs Number of Robots 76

List of Tables

Table 2.1 Priority of Checking and Prefer Directions.....16

Table 2.2 Description of Each Protocol 26

Abstract

Wireless sensor and robot networks (WSRNs) are an integration of wireless sensor network (WSNs) and multi-robot systems. They comprise of networked sensor and mobile robots that communicate via wireless links to perform distributed sensing and actuation tasks in a region of interest (ROI). In addition to gathering and reporting data from the environment, sensors may also report failures of neighboring sensors or lack of coverage in certain neighbourhood to nearby mobile robot. Once an event has been detected, robots coordinate with each other to make a decision on the most appropriate way to perform the action. Coverage can be established and improved in different ways in wireless sensor and robot networks. Initial random sensor placement, if applied, may be improved via robot-assisted sensor relocation or additional placement. One or more robots may carry sensors and move within the ROI; while traveling, they drop sensors at proper positions to construct desired coverage. Robots may relocate and place spare sensors according to certain energy optimality criteria.

This thesis proposes a solution, which we call Election-Based Deployment (EBD), for simultaneous sensor deployment and coverage maintenance in multi-robot scenario in failure-prone environment. To our knowledge, it is the first carrier-based localized algorithm that is able to achieve 100% coverage of the ROI with multiple robots in failure-prone environment since it combines both sensor deployment and coverage maintenance process. We can observe from the simulation results that EBD outperforms the existing algorithms and balances the workload of robots while reducing the communication overhead to a great extent.

Chapter 1: Introduction

1.1 Background Information

Sensors are used to measure physical features and provide a corresponding electrical signal which can be read by electronic equipment. They can be divided into different types according to the conditions they monitor, such as temperature, humidity, pressure and light. A WSN is generally composed of a large amount of distributed sensor nodes, with each one having limited and similar sensing and communication capability [1], [2], [3]. A WSN gathers information in a specific area by covering the whole region in sensors sensing range. It aims to collect accurate data which can reflect the real condition of the monitored area, which is the ROI. However, sensing holes caused by unbalanced formation or sensor failures after long-term usage will affect the accuracy of information captured by the network. Thus, the coverage ratio is an important metric for evaluating the performance and capability of the WSN.

There are two main sensor deployment approaches, which can establish a sensor network on an empty, unknown and hazardous ROI: carrier-based deployment and sensor self-deployment. The former loads sensors on robots, such as robots, to deploy them on an expected target or a map of the whole ROI set in advance [1]. However, the latter drops mobile sensors randomly from a safe location, such as aeroplane, and the sensors deploy themselves to achieve an optimal coverage afterwards [4].

Once a network is initially established, it should be able to work normally for a desirable time period. For a sensor deploying system, robots changes their roles from deployer to maintainer, which aims to maintain the sensor network by detecting and patching sensing holes. By comparison, in a system without robots, sensors have to relocate themselves to cover sensing holes and achieve optimal coverage.

1.2 Problem Statement

1.2.1 Sensor Placement

Carrier-based sensor deployment aims to place static sensor nodes in a two-dimensional empty ROI with preliminary target positions in Wireless Sensor and Robot (also called Actuator or Actor) Network (WSRN) [5]. Each robot loads a sufficient amount of sensors and is able to move throughout the ROI to achieve full coverage. Robots are aware of the positions where they enter into the region and are able to detect physical obstacles and boundaries of the ROI. Sensors have limited but homogeneous sensing radii, which is r_S. Both sensors and robots have the same communication radius, which is r_C, and they are able to transmit messages within their communication ranges. With the formation of map set in advance, the main task for robots can be described as to fully cover the whole ROI by constructing a connected sensor network.

During the deployment, the energy and communication consumption should be minimized. Although sensors are designed with low energy consumption, they can only survive within a very limited lifetime with current technologies [1], [3], [6], [7], [8]. Thus, all these special characteristics of sensor network require the algorithm to reduce move steps, time and communication consumption.

1.2.2 Coverage Maintenance by Robots

Once all grid points in the ROI are fully occupied by sensors, we should make sure the whole system works normally. Since sensors have short lifetime, as previously stated, sensing holes resulting from sensor failure will lead to the inaccuracy of data. A brief description of the maintenance problem is given as follows.

All the robots are loaded with sufficient sensors which can work normally. They wait in the ROI in a “sleep” mode for the rise of failures. Each robot is able to move throughout the ROI, replace sensors and communicate with other devices within its communication range. While sensors can detect failures of their neighbors and send failure reports, robots should be able to move to the sensing hole immediately and

replace the failed sensors with new ones once they receive a failure report.

Since the time to patch sensing holes should be minimized to guarantee the accuracy of information, the time consumption should be taken into consideration in algorithm design besides the energy and communication.

1.3 Existing Solutions

1.3.1 Sensor Placement

An important issue of carrier-based sensor deployment can be described as how to guide robots to achieve full coverage within the shortest time and lowest energy and communication consumption. The existing algorithms about carrier-based deployment have several limitations, which will be reviewed briefly in this chapter.

Batalin and Sukhatme proposed the Least Recently Visited approach (LRV) [9] to solve the problem in single robot scenario, which is not practical for large scaled network. It is high energy and communication consumptive and does not have a clear terminating condition.

Chang et al. proposed Obstacle-Resistant Robot Deployment approach (ORRD) [10], which guides robots to move like snakes. However, it does not guarantee full coverage in some specific conditions even in failure-free scenario. Furthermore, the authors did not mention under which condition the algorithm will terminate.

Back-Tracking Deployment approach (BTD) is an algorithm proposed by Li et al. [11], [5], to resolve deployment problem over the ROI for both single-robot and multi-robot scenario. It performs well in failure-free environment, whereas it cannot guarantee full coverage in failure-prone environment.

We can see each of these methods have major deficiencies such as inefficient movement and communication, unclear terminating condition or lack of full coverage guarantee.

1.3.2 Coverage Maintenance

Mei et al. [13] proposed a cluster-based approach to maintain the network in a failure-prone environment. Three protocols are proposed for different scenarios,

which are centralized manager algorithm, fixed distributed manager algorithm and dynamic distributed manager algorithm. However, several drawbacks and unclear descriptions exist in these protocols, such as redundant messages, unclear region division and communication interrupting. All the approaches listed above will be reviewed in detail in the next chapter.

1.4 Motivations and Objectives

Considering the three mentioned algorithms are the only proposed localized solutions to the carrier-based sensor deployment problem, and none of them guarantees full coverage in failure-prone environment since they do not have a corresponding coverage maintenance algorithm to help them support the network after initial covering, a localized solution which combining deployment and maintenance for multi-robot scenario is needed. The reason of choosing localized algorithm is that it is able to minimize communication consumption and deal with sensor failures in multi-robot scenario [14].

We designed a localized sensor deployment algorithm extended from BTD and proposed a new coverage maintenance approach corresponding to it to solve the carrier-based sensor deployment problem and network maintenance problem after that. These two algorithms work together as a whole, which is called Election-Based Deployment (EBD). The proposed algorithm is expected to outperform the existing solutions in energy and communication consumption and hole patching latency, while providing coverage guarantee and network support in a failure-prone scenario.

1.5 Assumptions

In the proposed algorithm, the WSAN is composed of mobile robots and static homogeneous sensors, with both having the same communication radii. The sensing ranges of sensors are of the same size, which is less than a half of the communication radii. Each device is able to communicate directly with others in its communication range without transmission collision and failure. All the established message transmitting relationships can be considered as bidirectional.

Each robot is aware of the starting locations of others in the ROI and is small enough to move through the interval of sensors. Robots are able to load sufficient number of sensors during deployment. In coverage maintenance, the number of sensors they can pick up is unlimited. Furthermore, robots have enough energy to move throughout the ROI to achieve full coverage and maintain the network afterwards.

1.6 Contribution

We proposed a novel algorithm, named Election-Based Deployment, which combines localized sensor deployment and localized network maintenance to achieve full coverage for multi-robot scenario in failure-prone environment. The deployment section is extended from BTD approach for a higher working efficiency by allocating different strategies to actuators according to the starting positions and modifying the backtracking protocol at the dead end situation. The hole fixing section aims to minimize the latency of hole covering by imitating election process. It selects the optimal robot to patch a hole by choosing candidates firstly and then electing a 'worker' among them to fix the sensing hole.

EBD is able to place sensors in an empty ROI and maintain the network subsequently, which is a new combined algorithm achieving both goals. It is low energy and communication consumptive due to its localized characteristic, and guarantees full coverage as well. Furthermore, it minimizes the latency of hole patching, which improves the accuracy of data gathered by the WSN in failure-prone scenarios.

1.7 Organization of the Thesis

The remainder of the thesis is organized as follows: In Chapter 2 three sensor deployment algorithms and a coverage maintenance method are reviewed. In Chapter 3, the EBD approach is discussed in detail while the simulation results are presented in Chapter 4, followed by conclusion and future work in Chapter 5.

Chapter 2: Literature Review

Sensors are used to measure physical features of the ROI. They transmit physical qualities like temperature, humidity or pressure to electrical signals. Sensors can be placed in a variety of ways. They can be deployed by robots in the WSRN, which are carried by robots and dropped on proper positions to meet a high coverage ratio. On the other hand, sensors can also be deployed by themselves, since mobile sensors can also move automatically after they are dropped in the ROI. They modify their original positions for the purpose of improving existing coverage, which is known as sensor self-deployment.

In this chapter, some fundamental concepts of self-deployment will be briefly discussed, three carried-based sensor deployment approaches will be introduced, and a brief description of a sensing hole patching algorithm will be given as well.

2.1 Sensor Self-Deployment

Sensor self-deployment is a sensor deployment method dealing with autonomous sensor coverage in mobile sensor networks (MSN). It can be used to place mobile sensors which can deploy themselves in the working area without robots. Because sometimes the environment is large scaled, unknown and full of unpredictable events which may cause sudden failure of sensors, such as volcanoes, deserts [15], [16], [17], [18], it is impossible to throw sensor nodes in expected targets or provide a map of the whole area in advance [1]. In this scenario, sensors should be able to adjust their position to reach a desired coverage before monitoring the environment [19].

In sensor self-deployment, sensing devices are dropped in the ROI randomly by an aircraft from a safe location, so the initial employment does not guarantee full coverage and uniform sensor distribution over the area [4]. To meet a higher coverage ratio, once sensors are dropped on the ground, they will search for an ideal position for themselves immediately.

Although sensors are designed with low energy consumption, they can survive for only a very short lifetime with current technologies [3], [11], [12], [13]. Due to the limited power availability with each sensor, energy consumption is a primary issue when designing self-deployment scheme for mobile sensors [4]. Furthermore, the low computing capability, limited memory and bandwidth of the sensors prohibit the working effects of some high complexity self-deployment algorithms [1].

There are eight common self-deployment approaches which have been proposed on sensor self-deployment issue: virtual force (vector-based) approach, Voronoi-based approach, load-balancing approach, stochastic approach, point-coverage approach, incremental approach, maximum-flow approach and genetic-algorithm approach. The first five of them are distributed or localized approaches whereas the rest are centralized approaches requiring a global view of the whole network [12]. The self-deployment is becoming an active research subject that draws the attention of an increasing number of researchers.

2.2 Sensor Placement by Robots

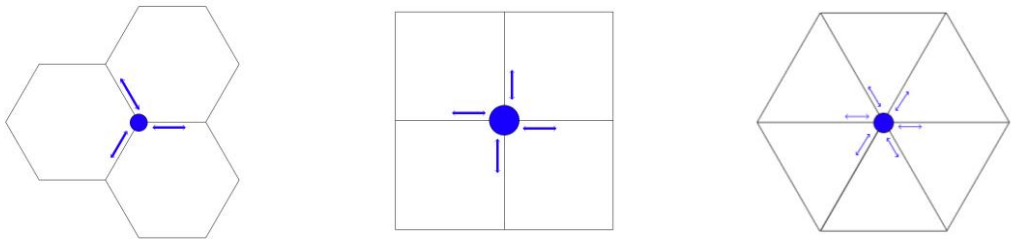
Sometimes, sensors are deployed by robots. They are carried by one or more than one robots. These robots will drop sensors while they are moving around in the ROI. Most of time, robots are assigned with specific routes and they will drop sensors following a grid which has already been assigned to guarantee the full coverage of the whole ROI.

Deploying sensors by robots is still an active research subject which is continuously drawing large amount of attention. There are only a few algorithms proposed to address this problem till now. Three mainly used approaches will be introduced in detail in the following sections. They are Least Recently Visited approach (LRV), Obstacle-Resistant Robot Deployment approach (ORRD) and Back-Tracking Deployment approach (BTD).

2.2.1 Least Recently Visited Approach

Least recently visited approach (LRV) is presented by Batalin and Sukhatme [9] to solve the sensor placement problem in a single robot environment. The sensing and communication radius of each sensor in LRV is equal to each other because all of the sensors are assumed the same. The robot starts with an empty environment from the outset.

In LRV approach, each node has several directions which represent the geographical directions the robot is allowed to move to. The number of directions of each node is equal and independent to each other. Figure 2.1 gives us examples of one node with three, four and six directions. In LRV, the amount of directions each node holds is four. Thus, the robot moves following a square grid and deploys sensors on the cross of two perpendicular lines.



(a) Three directions (b) Four directions (c) Six directions

Figure 2.1 Nodes with Different Directions

In LRV, i is used to represent nodes, for every node i, $D(i)$ is the set of directions this node holds, which allows robot to move close to or away from i. Another important definition in LRV is weight, which is represented by $W(i,d)$. Then $\forall d \in D(i)$, $W(i,d)$ represents the time that direction d has been traversed to or away from. Every time the robot passes by node i, $W(i,d)$ increases by one before direction d or after d is traversed.

At initiation, the robot deploys a sensor at the starting node, which can be considered as its current position. Every time a new node is placed, it should have connection with at least one deployed sensor in the whole network, except the first

placed sensor. After the first sensor is deployed on the starting node, there is no sensor in its communication range.

Then, the robot has to make a decision as to which direction to move so as to place the next sensor. It will choose the locally least recently visited direction to move to, which is the one with the smallest weight W. The robot receives a message which includes the least visited direction from its current sensor when it is still in the communication range of this sensor. If there are multiple directions which hold the smallest weight, the robot will make a selection according to a predefined order, which is South, East, North and West. If the chosen direction is obstructed by a boundary or an obstacle, the robot will inform the message sender and ask for a new suggested direction. Once the sensor receives the message, it will mark the previously recommended direction as an obstructed direction, and suggest a new one.

After the robot moves to a new node, it is unknown for it if this node has already been occupied with a sensor. It will wait for a certain short period of time which has been predefined. If it does not receive any message from a sensor, it will regard this node as an empty node and drop a sensor there.

Figure 2.2 is an illustration of the LRV approach. The blue circles in the graph represent sensors while the number in each circle means the sequential number of the sensor. The four numbers around the circle symbolize weight of four geographic directions, cross is used to mark obstructed direction, which may lead to a boundary of the environment or an obstacle.

A robot starts from node A and intends to move to one direction to deploy another sensor. Before it moves to the second node, the weight of four directions are zeros, as shown in Figure 2.2(a). Then it travels towards south, which is the highest order of the four geographic directions. Because it moves from north to south, the weight of south of Sensor 1 and north of Sensor 2 update to one, shown in Figure 2.2(b). Then the robot keeps placing sensors until arriving at the southwest corner. Sensor 4 firstly recommends south to the robot, however, the robot reaches the boundary of the environment after it moves to south. So it sends a message back to

Sensor 4 asking for another recommendable direction. This time east is chosen, thus the robot moves towards east and drops Sensor 5, as shown in Figure 2.2(c).

However, the LRV has an obvious drawback, which is illustrated in Figure 2.2(d). After the robot places Sensor 16, the weight of three directions are zeros except the incoming direction-north. It moves to the south firstly and waits there for a short period of time. If there is no message sent from any sensor during this period, the robot will consider the node as empty and drop a sensor there. However, it receives a piece of message from Sensor 7, which forces the robot to go back to Sensor 16. Then the same situation happens again, the robot moves east before finally moving to proper direction-west and places Sensor 17. One of the disadvantages of LRV is shown here, after the robot drops Sensor 16, there is only one empty node among the adjacent nodes of sensor 16. However, it takes extra time and redundant movement for the robot to travel to other two directions and go back.

Then the robot keeps traveling until it gets stuck at Sensor 22. It follows the recommendations of Sensor 22 and 21 and finally moves to the right direction and places Sensor 23 at the northwest corner. With the Sensor 23 deployed, the whole environment gets fully covered.

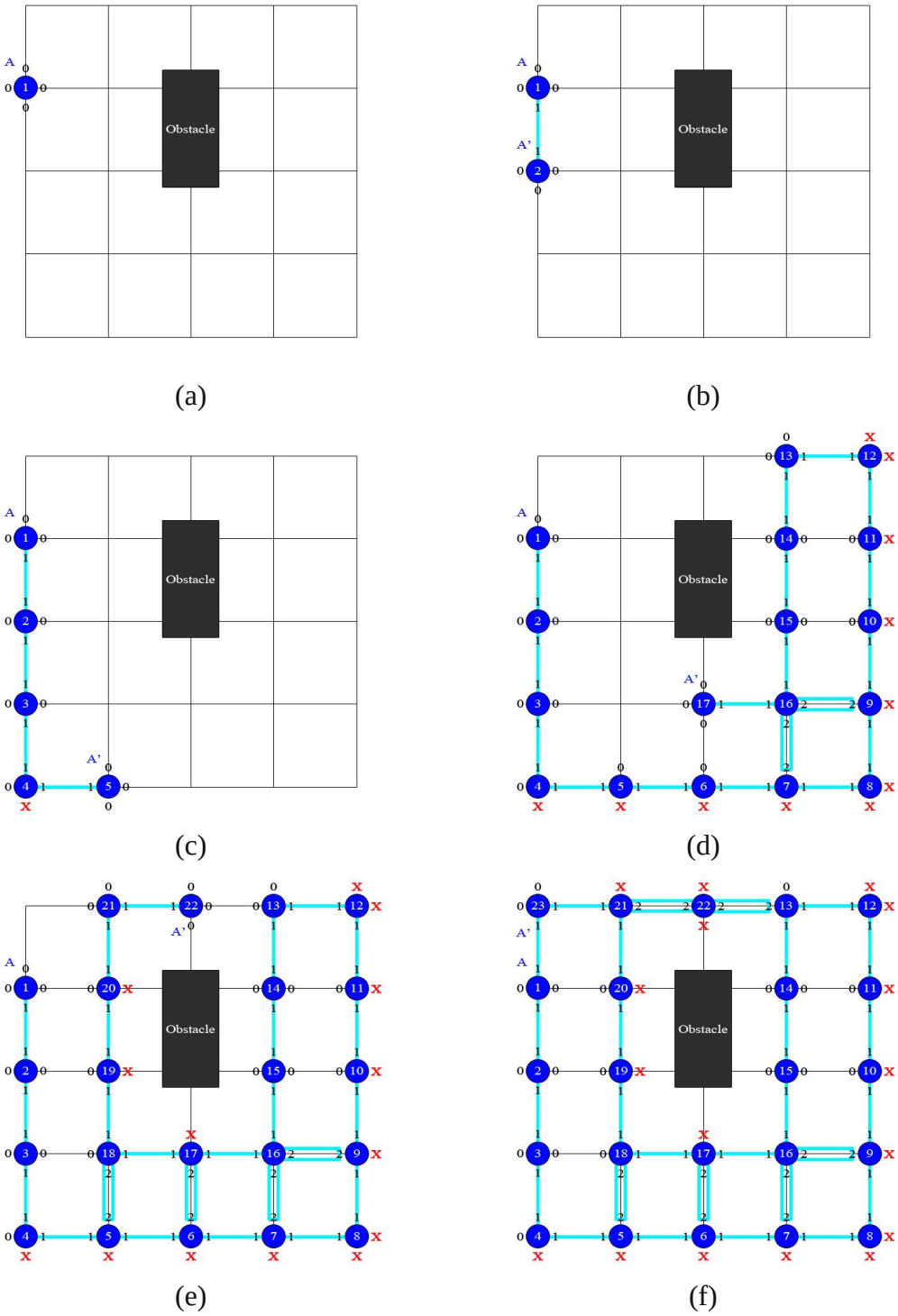


Figure 2.2 LRV Approach

Then there is another problem that comes with the full coverage. The author did not mention under what situation the approach terminates. Because of lack of a global view of the circumstance of coverage, the robot does not know whether the environment is fully covered or not. The robot will not stop until there is no sensor

left in hand. However, on account of the exhaustive movement, the existing sensing holes caused by runtime node failure are more likely to be visited and patched by the robot.

In the paper [22], the authors mention two types of multi-robot extensions of the LRV algorithm. The first one is to divide the robots into two groups, which can be distinguished by their tasks during the deployment. The two groups are transporters and builders respectively. Transporters are in-charge of delivering the sensors to several piles, whereas builders take the responsibility to pick the sensor and deploy them in the environment. Another extension is to let a team of robots working on coverage together, which is much more like what we do in this article. During the deployment, robots exchange information about their local positions and update their conditions through the deployed network. However, both of the two extensions are considered as a future work and the authors did not give further details about them.

2.2.2 Obstacle-Resistant Robot Deployment Approach

Obstacle-resistant robot deployment approach (ORRD) is proposed by Chang et al. [10]. It is based on the Obstacle-free robot deployment approach (OFRD) presented by Chang et al. [23], which is another algorithm which aims to address sensor deployment in a single mobile robot scenario.

The only robot places sensors following a triangle tessellation, which is similar to the grid in which every node has six moving directions in the last section. The triangle tessellation is illustrated in Figure 2.3. The black circle in the graph represents the sensing range of each sensor, while the black point is used to mark the position where the sensor has been deployed. The nodes where sensors are placed form an equilateral triangle, hence the triangle tessellation.

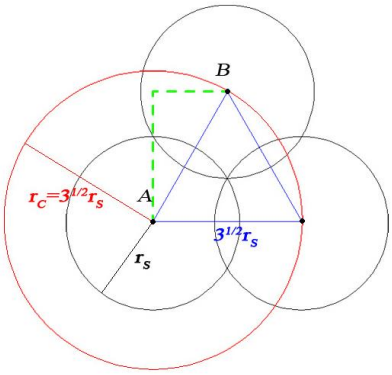


Figure 2.3 Triangle Tessellation

Since every sensor should be able to communicate with the sensors adjacent to it, the radii of communication range which is illustrated by the red circle should be no smaller than that of $\sqrt{3}$ times of sensing range. Let r_S and r_C denote the radius of sensing and communication ranges respectively, r_S and r_C should satisfy Equation (2.1).

$$r_C \geq \sqrt{3} r_S \tag{2.1}$$

The difference between the six-direction grid in Section 2.1.1 and triangle tessellation is that, in OFRD approach the robot cannot move to six directions. The robot is only able to move towards four fundamental geographic directions from each node: left, right, up and down. For example, if the robot intends to travel from node A to B in Figure 2.3, it will not move along the side of the triangle from A to B directly, instead, the robot will move towards North and East for $\frac{3}{2} r_S$ and $\frac{\sqrt{3}}{2} r_S$ respectively, as shown by the green dash line in Figure 2.3.

The authors assigned the robot to move along horizontal line and deploy sensors at separation $\sqrt{3} r_S$ until it reaches left or right boundary of the monitoring region or an obstacle. Then it moves $\frac{\sqrt{3}}{2} r_S$ down to the next horizontal line and keep deploying sensors to an opposite moving direction, and proceed similarly, as shown in Figure 2.4.

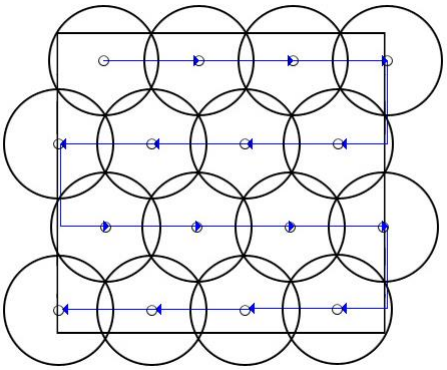
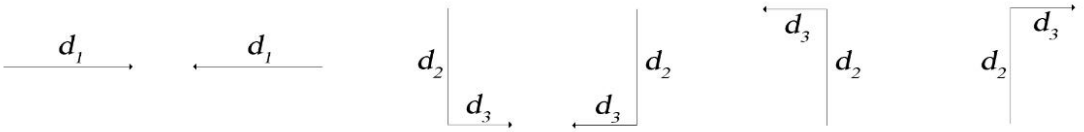


Figure 2.4 OFRD Deployment

Since the robot has only limited types of movement from one node to its adjacent one in triangle tessellation, six types of basic movements are proposed, which contains all of the possible movement steps in OFRD approach. They are shown in Figure 2.5.



(a) Type 1 (b) Type 2 (c) Type 3 (d) Type 4 (e) Type 5 (f) Type 6

$$d_1 = \sqrt{3} r_S, \quad d_2 = \frac{1}{2} r_S, \quad d_3 = \frac{\sqrt{3}}{2} r_S$$

Figure 2.5 Six Basic Movements

Since the robot will move horizontally no matter which type of movement it follows, two states are presented, which are East and West. These two states represent the robot is currently moving towards east and west respectively. The robot stays in one of these two possible states in the OFRD approach. For each state there are six directions, which can be divided into a set of Checking Directions and a set of Prefer Directions. In each set there are three directions respectively. The Prefer Directions are used to guide the robot to a proper direction of the next movement, whereas the Checking Directions are proposed for the robot to further check if any sensing hole existed next to its current position before its next movement.

There is priority order in each state of movement, which is listed in Table 2.1. The table can be considered as a rule which guides the robot to move in the OFRD approach.

Table 2.1 Priority of Checking and Prefer Directions

States	Checking Direction 1	Checking Direction 2		Prefer Direction 1	Prefer Direction 2	
East	Type 2	Type 5	Type 6	Type 1	Type 3	Type 4
West	Type 1	Type 6	Type 5	Type 2	Type 4	Type 3

The robot will check the three Checking Directions before every next movement step in order to avoid missing any sensing hole hidden behind obstacles. If a hole is detected, the robot will change its moving direction from the current one to the hole and deploy sensors in the hole first. If the answer is negative, the robot will move along one of the Prefer Directions.

The Figure 2.6 depicts movement of robot in a scenario with an obstacle. The black block represents the obstacle and the blue lines denote the trajectory of the robot movement by applying the OFRD approach. The larger circle is used to mark the sensing range of a sensor while the small circle in it symbolizes the location of sensor. The gray area in Figure 2.6(c) represents a sensing hole which has not yet been covered by sensors.

The robot starts from the left top corner in this example, as shown in Figure 2.6(a). However, since the approach involves the consideration of the existence of obstacles and sensing holes, it allows the robot to start from any corner or the middle of the monitoring region.

The robot stays in the East state and after placing sensor A, it reaches the left boundary of the monitoring region. So it changes its movement from Type 1 to Type 4 so as to deploy sensor B on the next horizontal line, as shown in Figure 2.6(b).

Figure 2.6(c) exhibits the obstacle-free robot deployment. After the robot deploys sensor C, there is an obstacle on its coming direction. Since the robot is in the East state, it checks Checking Direction 1 (West), but there is a deployed sensor in the West direction, the movement in Checking Direction 1 is failed. Then, the

robot further checks Checking Direction 2 (North) and no hole is found. So the robot moves to the Prefer Direction 1 (East), but the movement is failed again because of the obstacle. Finally it moves in Prefer Direction 2 (North) bypassing the obstacle and places the sensor D.

After deploying sensor D, the robot still stays in East state, it checks Checking Direction 1 (West) first. This time, a hole is detected in the West direction. Then the robot moves with Type 2 movement and drops sensor E there. Finally, the robot overcomes the impact of obstacle and achieves the goal of full coverage deployment, as shown in Figure 2.6(d).

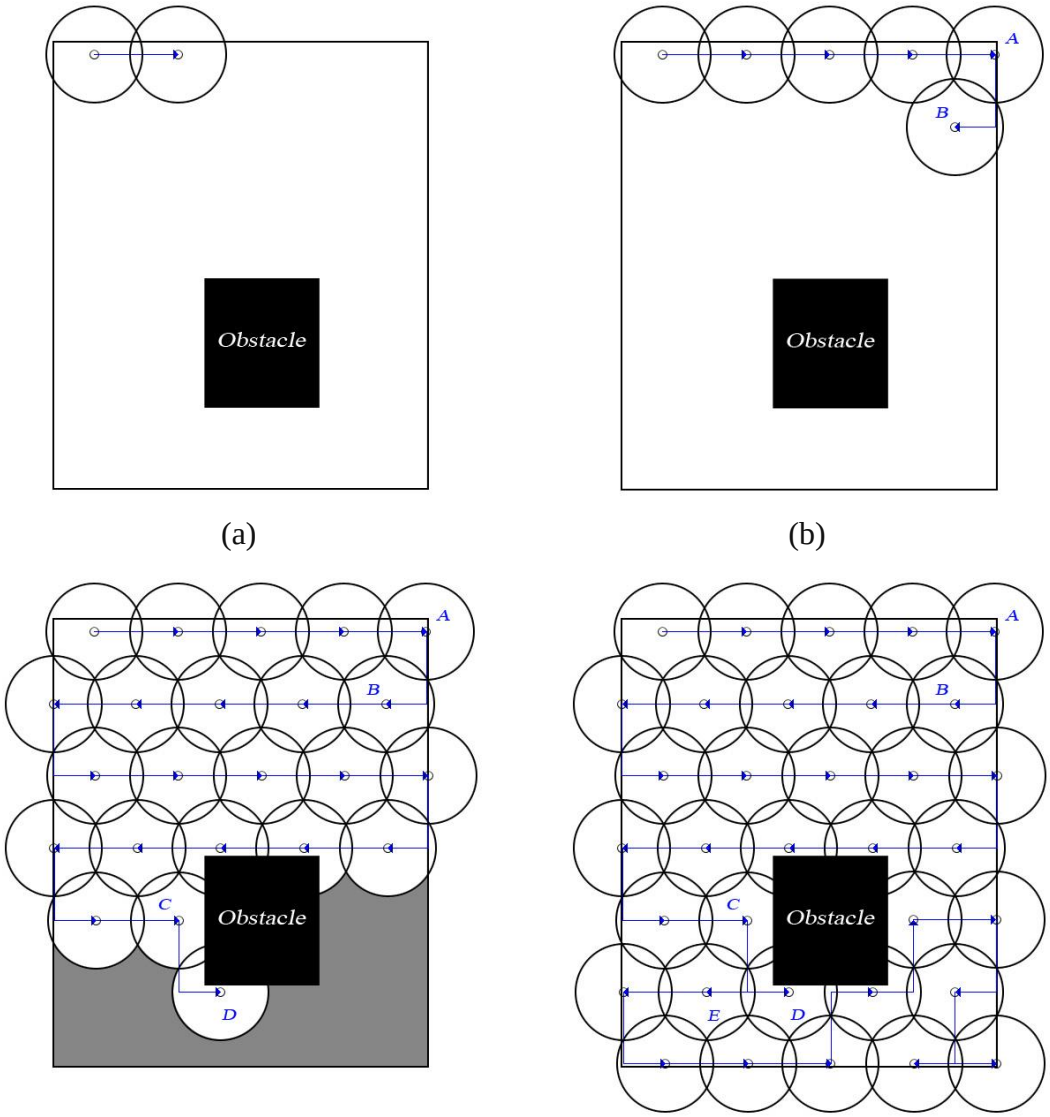


Figure 2.6 OFRD Approach in Obstacle Scenario

Obstacle-resistant robot deployment approach (ORRD) is proposed by extending the OFRD approach. It strengthens the ability of the algorithm to handle the boundary and obstacle problem. According to the authors, the robot will deploy sensors near the boundary or obstacles if lack of this sensor may cause a sensing hole.

However, it is also not clear that under what condition this approach terminates in both OFRD and ORRD approach. Unlike what the authors claimed, the algorithm in fact does not guarantee full coverage in some circumstance according to their current algorithm description. A simple counter-example scenario is designed by extending the example in Figure 2.6, which lengthens the obstacle until one of its end attaching the border of the monitoring region, as shown in Figure 2.7. In this situation, once the robot enters one side of the obstacle, it will not be able to enter the other side and leave an untreated hole.

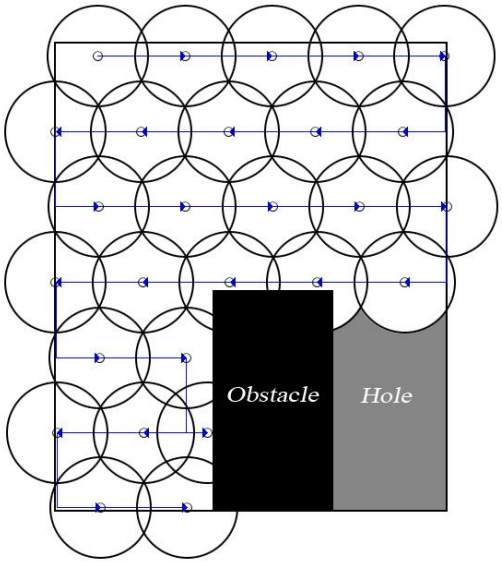


Figure 2.7 Incomplete Coverage by ORRD Approach

2.2.3 Back-Tracing Deployment Approach

Back tracking deployment approach (BTD) is an algorithm proposed by Li et al. [11], to resolve deployment problem over the ROI for both single-robot and multi-robot scenario. Four geographic directions are assigned: West, East, North and South, and

the robots drop sensors on a square grid. Robots check directions following the order West > East > North > South before they choose the moving direction of their next step, and as a result, robots can move in a snake-like way. Unlike LRV or ORRD algorithms, BTD approach terminates within finite deploying time and the ROI can be fully covered in failure-free environment.

Each sensor is dynamically assigned a color according to the condition of its neighbors. A sensor will color itself "*white*" if at least one of its one hop neighbor spots has not been placed a sensor yet, or "*black*" if all of its four neighbors are deployed with sensors working normally. Besides the color, every sensor stores a sequential number which represents the order of deployment and ID of the robot the sensor was dropped by. Sensors share these information with their one hop neighbors as well by messages.

Robots are able to detect not only their own location but also physical obstacles, boundary of the ROI and sensors which have been already deployed. Each robot has its own distinct ID and moves independently and asynchronously. It travels in the ROI according to a specific strategy and drops a sensor after every step if the spot is empty.

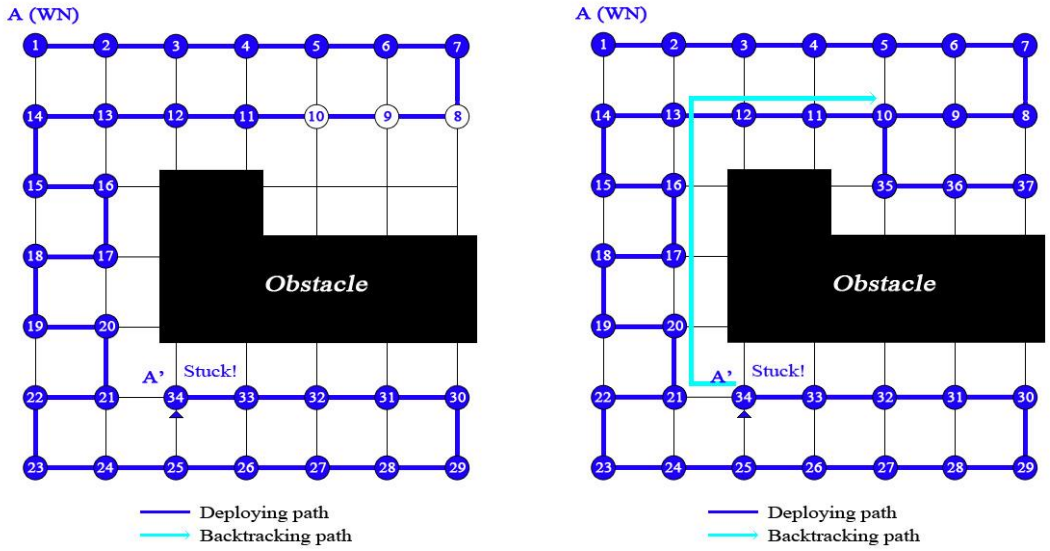
In BTD approach, sensors are deployed on the intersections of a square grid, and all grid points are initially empty. Each deployed sensor will send a message which carries its location and other necessary information, such as color and back pointer, to its one hop neighbor periodically. The message is a useful tool for sensors to know the condition of their neighbors. If a sensor receives messages from one of its neighbors periodically, it will mark this neighbor "normal". If a sensor never received any messages from a neighbor, it will mark this neighbor as "Empty".

In addition, there is a forward pointer and a back pointer stored in every sensor, which point to the successor and the last white predecessor respectively. The last white predecessor refers to first white sensor along the backward path of robot it was dropped by. In BTD algorithm, the color and pointers of each sensor are defined locally. Both of these pointers serve as navigation tool and help robots find its way when it gets stuck in a dead end situation.

A robot will meet a dead end if all of the one hop neighbors of its current position are occupied with sensors. When a robot reaches a dead end, it will back track to the white sensor with the largest sequential number, which is the back pointer, among the sensors deployed by itself or by another robot, and resume deployment from there. If a robot cannot find a back pointer on its current sensor, it will search for a back pointer stored in its one hop neighbors. If there is at least one back pointer gets found, the robot will move to one of them randomly; otherwise, it will send searching messages to four geographical directions, the messages will reach the boundary of the ROI and travel around the border. If the messages find back pointers, it will return with their locations; otherwise, if no back pointer is found, the robot will terminate.

The method for robots to move from a dead end to the back pointer is named "shortcut" method [24], which can minimize the number of moving steps by setting the robots move to next one hop neighbor with the lowest sequence number and the same back pointer, until reaching the destination.

Figure 2.8 is an illustration of shortcut method used as a guide in BTD approach to find back pointer. The back pointer stored in a robot is the white sensor with the largest sequential number along the back path of this robot. Robot A gets stuck at Sensor 34 in Figure 2.8(a), the back pointer stored in A is sensor 10. There are three one hop neighbors that have the same back pointer as A, which are Sensor 21, 25 and 33. Robot A will move to Sensor 21 since the sequence number is the lowest. Following the shortcut principle, robot A backtracks to sensor 10 and resumes deploying from there, shown in Figure 2.8(b). The approach is used to guide robots through an ROI with obstacles with the shortest moving distance.



(a) (b)

Figure 2.8 "Shortcut" Approach in BTD

A white sensor will count the number of empty spots among its one hop neighbors automatically once it colors itself "white". This number will be stored in this white sensor as a property. If one robot considers itself as a back pointer, it will send a request message to the sensor before it marks this white sensor as its destination. The robot will move to the white sensor only when its request is granted. If the sensor receives and permits a request message, it will decrement the number of empty spots by one. When the number goes down to zero, which means all of its empty spots will be deployed by specific robots, the white sensor will not permit request any more. In this case, although it is still a white sensor, it will be considered as a black sensor, and the back pointer stored in the sensors along the forward path will be replaced by the next white sensor along the backward path. Robots may move from one dead end to another dead end due to the network asynchrony and delay of information transfer.

Figure 2.8 illustrates how BTD works in single and multiple robots scenarios. In Figure 2.8(a), A meets a dead end after deploying sensor 17, since all four directions are obstructed by deployed sensors and obstacle. The back pointer stored in sensor 17 is sensor 16, thus robot A decides to back track to sensor 16 and resume placement from there. Then it gets stuck again at sensor 35 and moved to the back

pointer sensor 32. The fully covered ROI of single robot scenario is given in Figure 2.8(b), whereas Figure 2.8(d) and (e) show the deployment in multi-robot scenario.

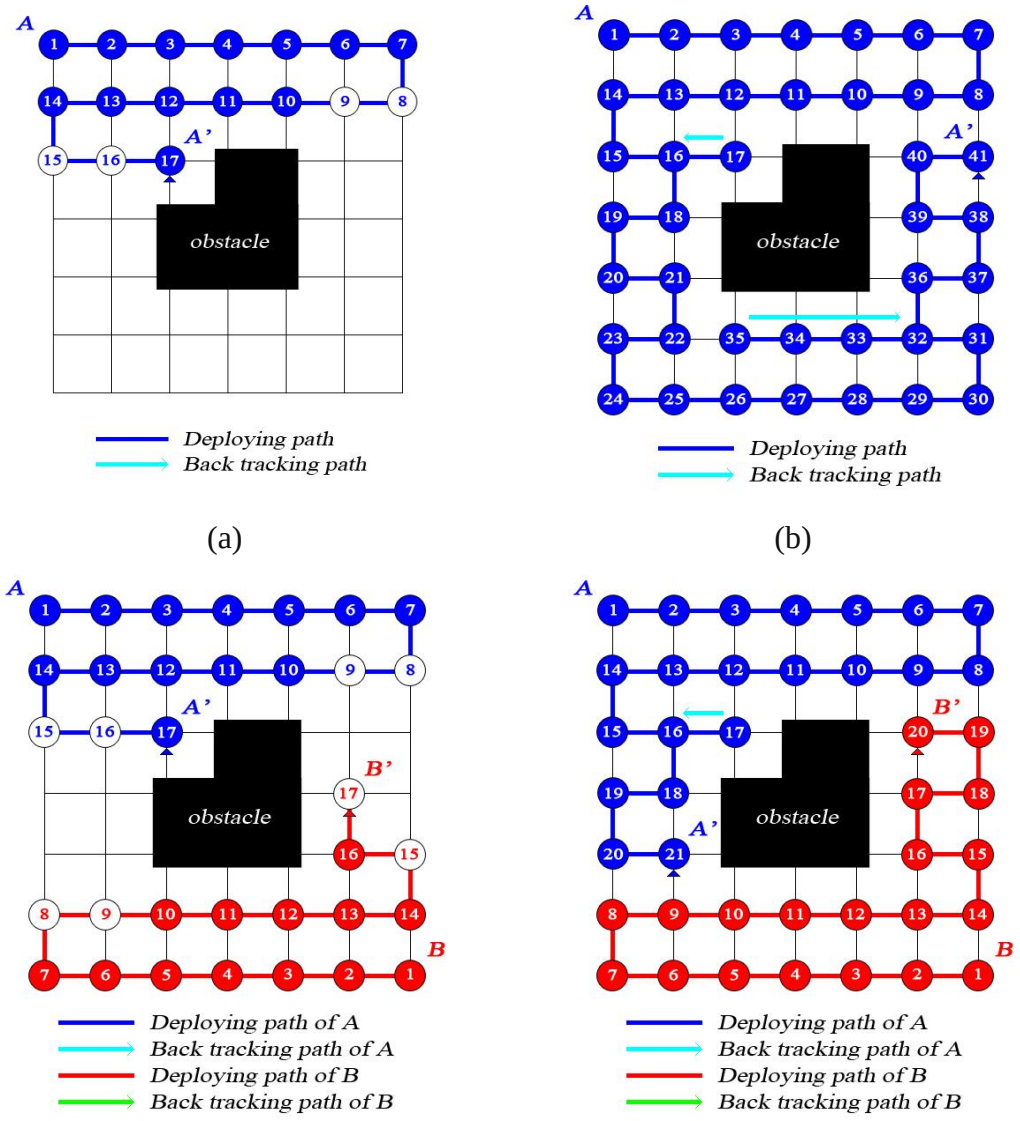


Figure 2.9 BTD Approach for Single and Multiple Robot Scenario

In reality, sensors cannot work for years without breakdown. Those sensors which are out of order will cause sensing holes in the ROI. Because of partial coverage, the accuracy of information measured cannot be precise. Also, the back pointer chain may be broken because of a sensing hole. In this situation, robots will be given another duty-sensing holes covering.

As we mentioned before, each deployed sensor will send a message to its one hop neighbor periodically. If a sensor receives messages sent from one neighbor at first, but cannot hear from it neighbor later. It is probably because of a breakdown occurs on this neighbor. In this situation, this sensor will wait until the deadline, if it still cannot receive any message from this neighbor, it will mark it with "failed". We know that sensors will be given different colors according to the state of their one hop neighbors. In this case, if a failed sensor existed among the neighbors of a sensor, it will color itself "*gray*". Therefore, gray sensors can be considered as a mark of position of failed sensors, since each of them is surrounded by a set of gray sensors, shown in Figure 2.10. Two failed sensors 26 and 31 are marked as black, their one hop neighbors then change their colors to gray.

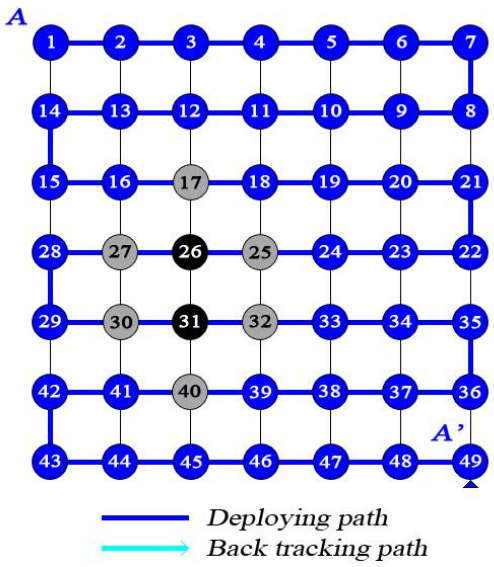


Figure 2.10 Sensing Hole Surrounded by Gray Sensors

In the failure-prone environment, robots follow the same strategies as they do in failure-free environment. Robots detect a sensing hole by encountering a gray sensor. They treat sensing holes like uncovered area and drop sensors there. Every time after a failed sensor is replaced, its one hop gray neighbors will change their color back to white. Robots will then resume deploying sensors from the hole after it is covered. However, robots are only able to patch sensing holes that they meet during backtracking. Sensing holes which do not affect backtracking will be left untreated.

For example, in Figure 2.11, robot gets stuck at the sensor on the northwest corner, whose sequential number is 72. The robot finds the backtracking path following the sequence 72->55->54->37->36->19->18->1 (black line with an arrow) until it gets to the back pointer, which is sensor 1, and resume deploying there. The black circles in the figure represent the failed sensors. We can see that sensors 17, 18, 19 and 20 form a sensing hole, which is hole 1; and sensor 41, 42, 49 and 50 form another sensing hole, which is hole 2. Hole 1 is on the backtracking path of the robot, which will be discovered and patched when the robot back tracks to sensor 1. However, hole 2, which does not affect backtracking, cannot be found and will be left untreated. In this scenario, BTD cannot guarantee full coverage in failure-prone scenario.

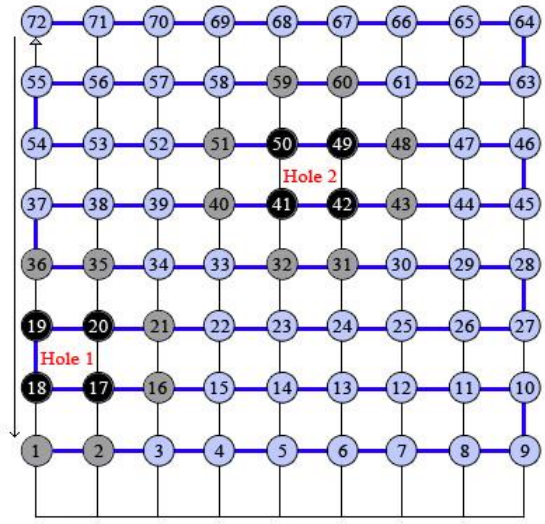


Figure 2.11 Drawback of BTD Approach

Moreover, load balancing is added to the algorithm, which balances the workload of different robots to avoid the circumstance that one robot keeps working whereas the others stuck in a dead end. BTD guarantees full coverage in failure-free environment, but it is unable to detect sensing holes which do not affect backtracking in failure-prone environment. Unlike LRV or ORRD algorithms, which do not contain conditions of termination, BTD approach terminates when there is no white sensor in the ROI.

2.3 Coverage Maintenance by Robots

After three robot-based sensor deployment approaches have been introduced, description of two hole fixing algorithms will be given. They are cluster-based and perimeter-based approaches. Both of these two approaches are proposed to patch holes caused by sensor failures in an already full covered area, and work under the assumption that robots are able to load sufficient number of sensors so that there is no need for robots to reload themselves with sensors.

Mei et al. [13] proposed a cluster-based approach to patch sensing holes by replacing failed sensors in the region. Three protocols are proposed for distinct scenarios-centralized manager algorithm, fixed distributed manager algorithm and dynamic distributed algorithm. In central approach, there is a robot that acts as the central manager which is in charge of receiving failure reports and then forwarding them to the responsible robot. However, in distributed approaches each robot can serve as the manager and maintainer in the meanwhile. Centralized and distributed algorithms perform differently in motion overhead and messaging overhead.

All the three protocols can be divided into three stages: initialization, failure detection and reporting, and failure handling. In initialization stage, each robot is assigned with a role, two sets of relationship are established, which are communication network between sensors and robots and network between guardians and the guardees. Initialization stage plays a role as the foundation in the whole protocol, sensor-robot and sensor-sensor communication networks are set up in this stage. Every sensor picks its nearest neighbor as its guardian after sending one message carrying its location to its neighbors and receiving one from every neighbor. Failures are detected by guardian of failed sensor and reported to a manager in the second stage. In the last stage, a maintainer robot is assigned with a task by the manager to patch the sensing hole.

For the purpose of comparison, description of each stage of each protocol is given in Table 2.2.

Table 2.2 Description of Each Protocol

Stage	Centralized Manager Algorithm	Fixed Distributed Manager Algorithm	Dynamic Distributed Manager Algorithm
Initialization	A manager is chosen. It broadcasts its location to all sensors and maintainers. Maintainers send their locations to the manager and one hop neighbor sensors. Sensors and robots communication network established. Sensors exchange messages with their neighbors and pick guardians, guardian-guardee relationship is set up.	The whole region is divided into several sub-areas and the number of sub-areas is the same as that of robots. Each robot can be considered as both manager and maintainer of that area. In each subarea, the sensor-robot network and guardian-guardee network are set up as centralized manager algorithm.	The whole region is divided into several sub-areas. Each sensor chooses its nearest robot as its manager. Sensors which have the same manager consist of the sub-area of this robot. Sub-areas may change with the movement of robots, as Voronoi graphs.
Failure detection and reporting	We assumed that guardee and guardian will not fail at the same time, because the probability is small and negligible. Sensors send message to their one hop neighbor periodically. If a sensor has not received message from its guardee for a threshold period of time, then it will consider this neighbor failed and send failure location to the manager.	Failure detection and reporting happens within the sub-area. The process of detecting and reporting is the same as that in Central Manager Algorithm.	Sensors report failures to their manager within the sub-areas, but the manager may change when robots are moving.

Failure handling	Once the manager receives a failure report, it will inform the closest robot as maintainer. The maintainer will keep updating its location while it is moving, because a moving maintainer can be assigned another task.	Once a robot receives a failure report, it will act as a maintainer, patching the hole by itself.	The procedure is the same as that in Fixed Distributed Manager Algorithm.

Some drawbacks of the protocols are listed here:

1. In failure detection and reporting stage, the same failure report may be sent for several times. Since no matter a guardee detects failure of its guardian or a guardian detects failure of its guardee, the detecting sensor will send a failure report anyway.

Figure 2.12 illustrates a scenario that the reports of the same failure are sent twice, which causes unnecessary message cost. There are six sensors and a robot in this WSAN. sensor B is the nearest sensor to A, so B is the guardian of A. While C is the nearest sensor to B and B is the same to C, thus B and C construct a guardian-guardee network. After the sensor network set up, if B breaks down, both A and C will detect the failure and send a report to the robot. Therefore, reports of the same failure have been sent twice. The authors did not mention what the manager should do if it receives several reports of one failure.

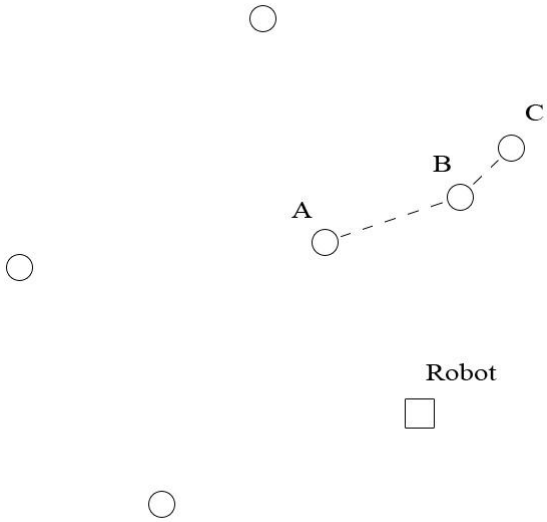
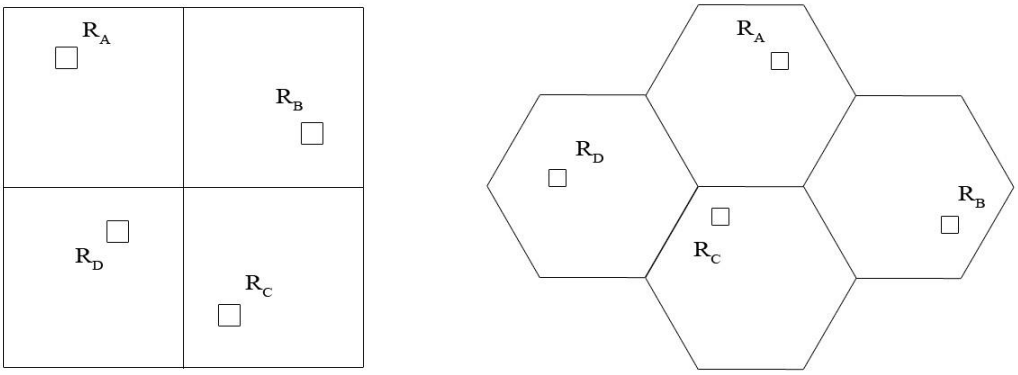


Figure 2.12 Message Resending

2. In the initialization stage of fixed distributed manager algorithm, the region should be divided into several sub-areas. However, it is not clear that how the region is divided. The authors give two examples of partition, shown in Figure 2.13, but it can only be used in specific situations. If the shape of the whole region is irregular, both of the squares and hexagons cannot be used.



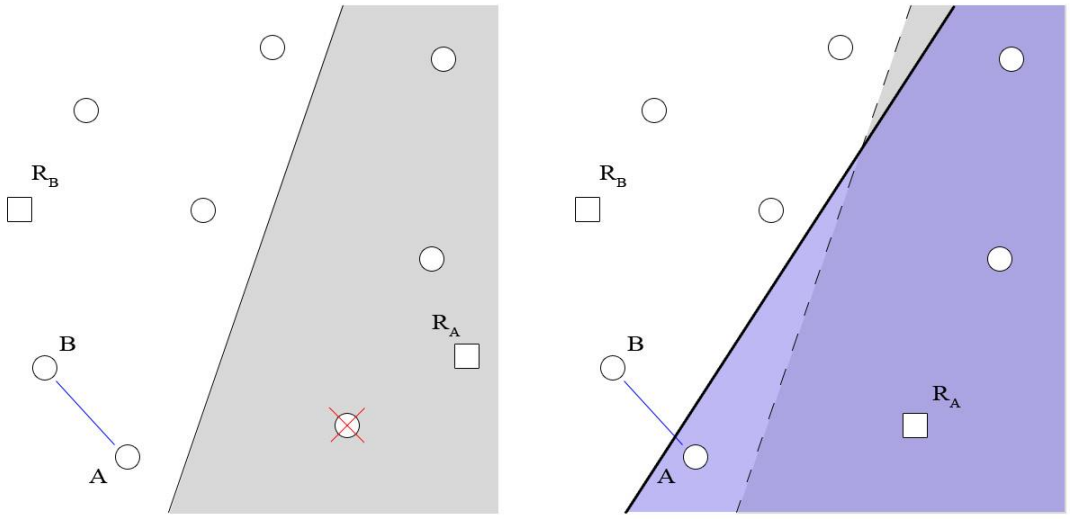
(a) Squares (b) Hexagons

Figure 2.13 Area Partition

3. In the last stage of dynamic distributed manager algorithm, there is no stationary boundary of sub-areas, a sensor may have several managers at different time. Thus, when the guardian-guardee network is built, the guardee and guardian are in the same sub-area for sure. However, with the robots movement, the guardee and guardian may change their manager and locate in two different subareas. It is probable that a sensor detects the failure of its guardee and sends it to its manager; however, this manager is not the manager of the failed sensor. As a result, a robot will have to travel to the subarea of another robot to patch the hole.

Figure 2.14 gives us an illustration about the break of guardian-guardee relationship caused by movement of robots. There are two robots in the figure; the gray region in Figure 2.14(a) represents the subarea of robot A, while the white is the one of robot B. Since sensor A and B are the closest sensor to each other, they form a guardian-guardee network connected with a short blue line. Everything keeps stationary until robot A receives a report of a failure in its area, which is marked with a red cross. After robot A has moved to the position of failed sensor, the boundary of two subareas changes to the bold black line shown in Figure 2.14(b). Sensor A is in the area of robot A now; if it fails, sensor B will send a report to robot B, although robot A is the responsible robot of sensor A at present.

To solve this problem, the guardian-guardee network has to be set up every time after a robot moves for a certain distance. However, updating the Voronoi diagram frequently will cost a huge amount of messages inevitably.



(a) (b)

Figure 2.14 Break of Guardian-Guardee Relationship

After comparison, the experiments show that the centralized manager algorithm and the dynamic distributed manager algorithm achieve a lower motion overhead, whereas the two distributed manager algorithms are more scalable and reach a high messaging overhead. However, all of these three protocols are expensive in message and energy requirements, and thus none of them is suitable for patching holes in large scale sensor networks.

Chapter 3: Election-Based Deployment Algorithm (EBD)

In this chapter, we will propose a new algorithm, which called Election-Based Deployment algorithm. The protocol contains two main phases: sensor deployment and coverage maintenance. In the first phase, robots place sensors in the empty ROI and terminate after all the grid points are occupied by sensors, which is extended from BTD algorithm in multi-robot scenario. In the second phase, the main task of robots is to patch sensing holes by replacing the failed sensors in order to maintain the network to work normally.

The first phase of our protocol will be introduced in Section 3.1, while the description of coverage maintenance phase will be given in Section 3.2.

3.1 Sensor Deployment

3.1.1 Strategy Allocation

3.1.1.1 Initial Strategy Allocation

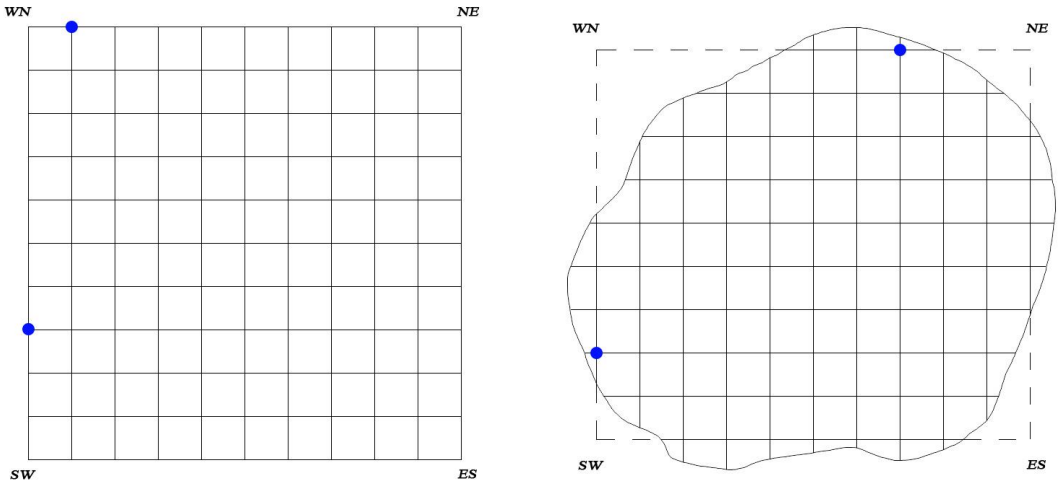
At the first beginning of deployment, each robot will be assigned with a moving strategy according to its starting position. Assuming the ROI is in a rectangular shape, we then have four moving strategies, each of which corresponds to one corner:

$$\begin{cases} WN : West > East > North > South \\ SW : South > North > West > East \\ ES : East > West > South > North \\ NE : North > South > East > West \end{cases}$$

"*WN*" represents moving strategy for robots whose starting positions are closed to the northwest corner. The robot assigned *WN* will check its west and east firstly and then its north and south before it moves a step. The situations of the "*SW*", "*ES*" and "*NE*" are the same.

If the ROI is rectangular, it is easy to find four corners, which are shown in Figure 3.1(a). There are two robots entering into the ROI from the blue points. It is clear that they follow strategies *WN* and *SW* respectively. However, for irregular ROI such as the one in Figure 3.1(b), the four corners can be found in the way shown in

the figure, although they may not really exist. The two robots follow strategies *NE* and *SW* in this example since the two blue points are closer to the northwest and southwest corners respectively.



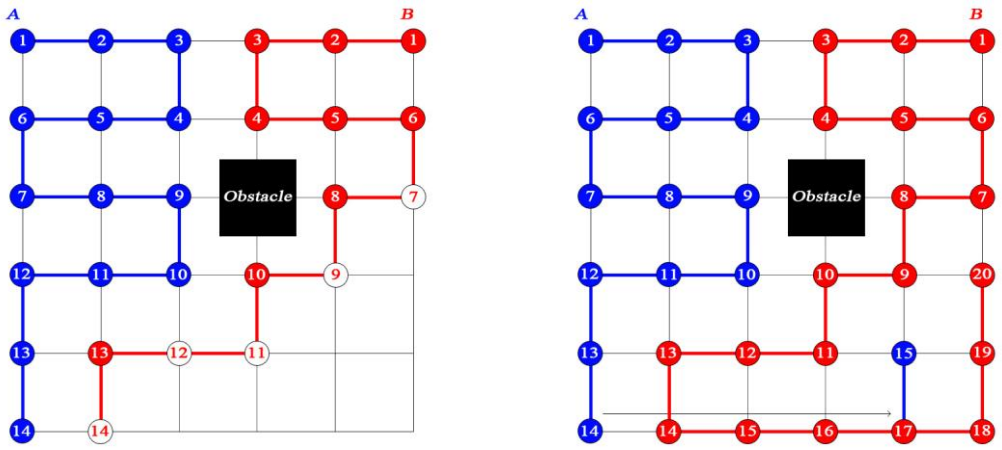
(a) Regular ROI (b) Irregular ROI

Figure 3.1 Initial Strategy Allocation

In our protocol, each robot knows the position of sensors it dropped because robots are aware of their starting coordinates when they enter in the ROI. Assumed the current location of a robot is (x_i, y_i), if the robot moves one step to west, the coordinate of the later dropped sensor will be $(x_i - 1, y_i)$. Similarly, we have $(x_i + 1, y_i)$, $(x_i, y_i + 1)$ and $(x_i, y_i - 1)$ for moving one step to east, north and south. After obtaining the coordinate, the robot stores all the position of sensors it deploys.

The reason for having different strategies is that strategy can be seen as a guide for robots when they are choosing moving directions. In the BTD approach, the priority is West > East > North > South. The robots starting from the eastern ROI may move to the west part when the eastern part has not been fully covered yet, which will result in a long backtracking path to the east after the west has been fully covered.

Take Figure 3.2 as an example. Robot A starts from the northwest corner and robot B starts from the northeast corner. We can see that after robot B deploys sensor 11, it will choose west because west has the highest priority in BTD algorithm, although the west part is supposed to be covered by robot A. Robot B moves into west part with the east part has not been finished yet. Figure 3.2(a) shows that after they finish deploying sensor 14, robot B moves back towards east to cover the last empty points, whereas robot A has to backtrack without deploying any sensor because it is one step after robot B. The black line with an arrow in Figure 3.2(b) represents the backtracking path of robot A. In this situation, working efficiency will be low since robot A does not deploy any sensor during backtracking.



(a) (b)

Figure 3.2 Drawback of BTD

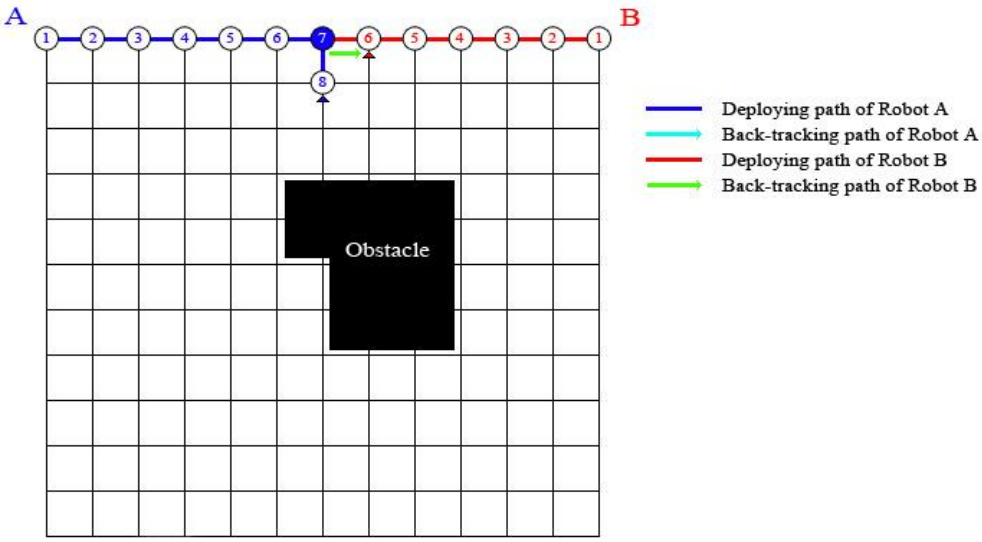
The following figures give us a clear comparison of the effects of single-strategy and multi-strategies in the same test environment of a two-robot scenario. Figure 3.3 shows the paths of two robots which follow the BTD approach, whereas in Figure 3.4, robots follow the new protocol, which assigns different moving strategies to these two robots.

In this thesis, we assign robot A moves first when two robots come across, robot A follows its strategy, while robot B considers the next point on its route to be filled with sensor already, and consequently changes its moving direction immediately to deploy sensors in another area. It is illustrated in Figure 3.3(a), two robots meet at

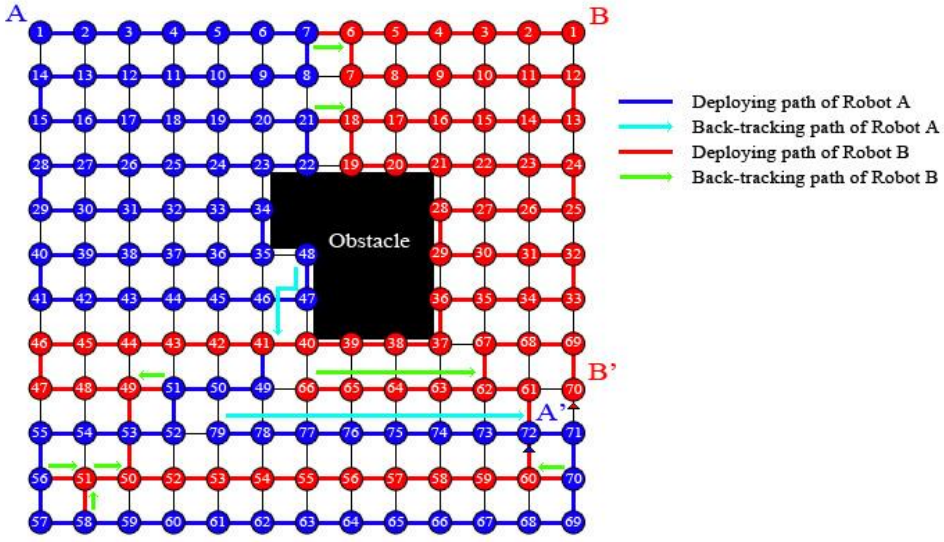
sensor 7, then A moves towards south to deploy sensor 8 while B backtracks to sensor 6, the sensors with triangles below represents the current position of robots.

It is clear that both BTD and EBD can guarantee full coverage in failure-free environment. However, robots travel a longer distance following BTD approach. It can be seen in Figure 3.3(b), which demonstrates a fully covered ROI. The light blue and green lines with arrows demonstrate the backtracking paths. Robot A deploys 79 sensors, while robot B deploys 70 sensors. Assuming the length of each square side equals to one, then the backtracking distance following BTD approach is 21 in total.

Compared with BTD, EBD guarantees a shorter moving and backtracking distance, which is shown in Figure 3.4. The sensors deployed by both robots consist two perfect triangles, which can be considered as a good beginning of the coverage maintenance phase. Also, robot A deploys 78 sensors and robot B places 71 sensors, which meets a more balanced workload. The distance of backtracking path is much shorter than that of the BTD approach, which is only 5.



(a) Encounter



(b) Full Coverage

Figure 3.3 Deployment of BTD

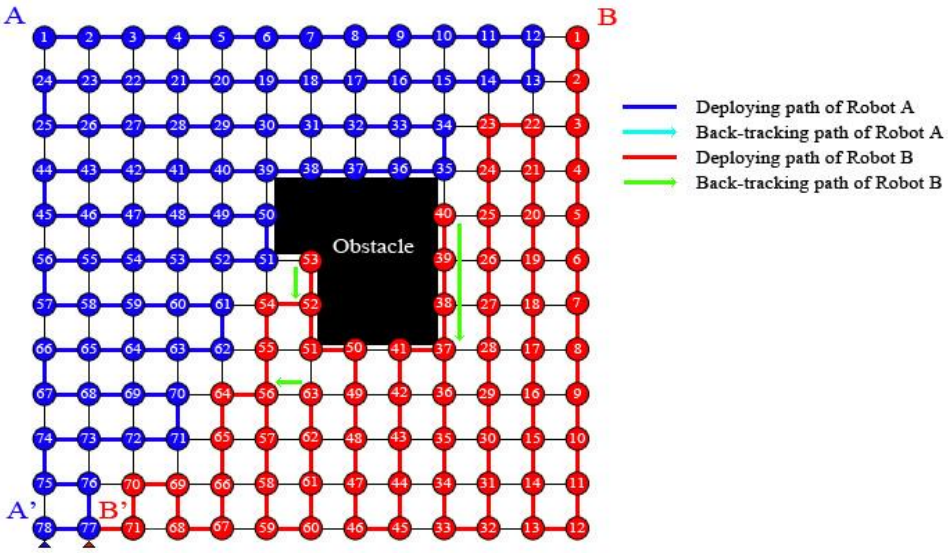


Figure 3.4 Deployment of EBD

3.1.1.2 Strategy Reallocation

The robots whose starting points are close to the same corner will be assigned to the same moving strategy. If two robots following the same moving strategy encounter, their areas will be crossed. Since the second section of EBD requires independent subareas, we have to avoid the case of several robots sharing the same moving strategy.

Take Figure 3.5 as an example. Robot A, B and C start from points closest to the northwest corner, so they are assigned to the *WN* strategy. Sensors deployed by the three robots form blue, red and yellow areas. We can observe the area of robot B, which is the red area, is cut into three pieces by the blue and yellow areas, the same as the subareas of robot A and C. Since in coverage maintenance algorithm, each robot takes responsibility to the sensors deployed by itself, the more regular the subareas are, the shorter distance robots move. Therefore, we need to reallocate the strategies in order to make subareas more regular in shape and independent to each other.

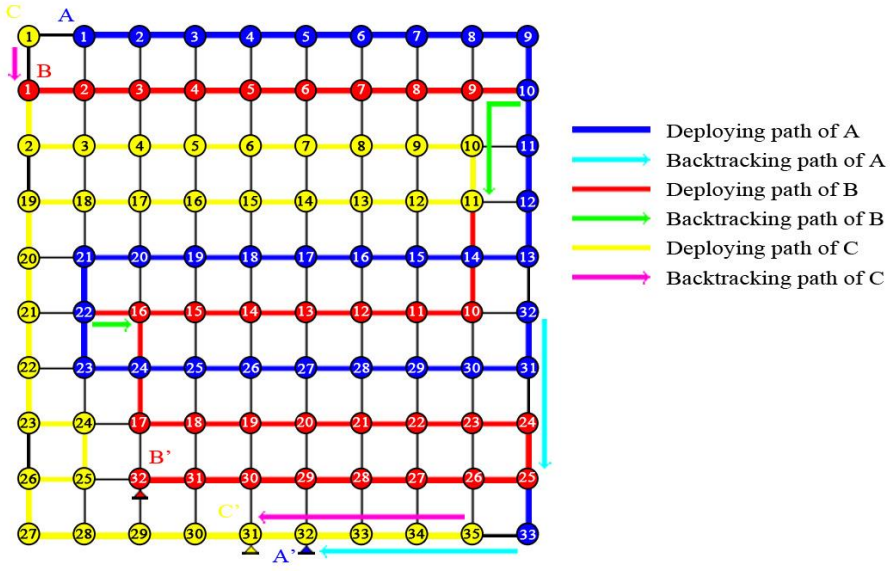


Figure 3.5 Crossed Areas

There are some definitions which should be introduced before we explain the details of strategy reallocation. First of all, the capacity of a strategy, which represents the number of robots follow this strategy. For example, if a moving strategy is assigned to three different robots, the capacity of this strategy is three. Another definition which should be clarified here is the "*empty*" strategy. It represents the strategy which is not assigned to any robot. On the contrary, if a strategy is assigned to more than one robot, it can be called "*rich*" strategy. The last one is "*neighbor*" strategy. If two corners are adjacent, we call the strategies corresponding to these two corners neighbor strategies to each other. For example, northwest corner and northeast corner are adjacent, thus strategy *WN* and *NE* are neighbor strategies.

Figure 3.6 shows the starting points of several robots, the black circles in the figure represent robots. We can see that, the capacity of strategy *WN*, *NE* and *SW* are 2, 1 and 3 respectively, while the capacity of strategy *ES* is 0, which means no robot starts from a point near the southeast corner. Thus, strategy *WN* and *SW* are rich strategies; whereas strategy *ES* can be considered as an empty strategy.

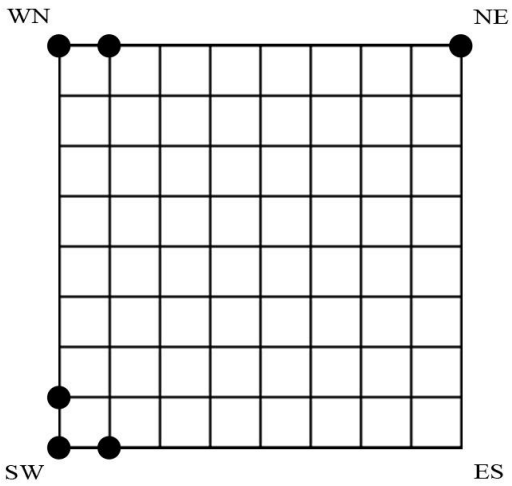


Figure 3.6 Rich and Empty Strategy

The target of strategy reallocation is to decrease the capacity of strategies. The reason for doing that is to decrease the number of robots sharing the same strategy. To solve this problem, we have to reallocate the moving strategies to robots when several robots sharing the same moving strategy, and this rich strategy has at least one empty neighbor. In this case, we will allocate the robot following the rich strategy to the empty one, until there is no empty or rich strategy.

Figure 3.7 gives us a simple example. We can see *WN* is a strategy assigned to two robots, *NE* is a single robot strategy, whereas *ES* and *SW* are empty strategies. Since *WN* and *SW* are neighbor strategies and *SW* is empty, we assign robot 1 to strategy *SW*.

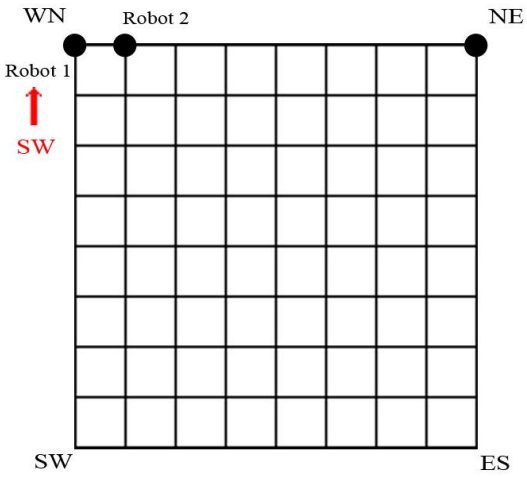


Figure 3.7 Strategy Reallocation

The example above is a simple one. If the scenario turns to be more complicated like the example in Figure 3.7, a reallocation algorithm is required to solve the problem. It is proposed as follows:

1. Checking the four strategies to find an empty strategy following the order *WN*> *NE*> *ES*> *SW*. If there is no empty strategy found among the four strategies, the reallocation algorithm will stop.

2. Once an empty strategy is found, the algorithm will search for rich strategies among its two neighbor strategies following the same order, *WN*> *NE*> *ES*> *SW*. There are three possible results:

(1) There is no rich strategy among the two neighbor strategies. The algorithm will go back to Step 1 and keep on searching for empty strategies.

(2) There is only one rich strategy among the two neighbor strategies. The algorithm will select the most suitable robot from all robots assigned to this rich strategy, following a picking algorithm which will be introduced later, when it chooses the most suitable robot, and this robot will be assigned to the new strategy.

(3) Both of the neighbor strategies are rich strategies. The algorithm will act the same way as it does in step (2). This time, the algorithm will select the most suitable robot from the robots that follow both of these two rich strategies.

3. After assigning the selected robot to the new strategy, the algorithm will resume searching for empty strategy until all four strategies have been checked.

A flow chart is shown in Figure 3.8, which illustrates the procedure of picking algorithm.

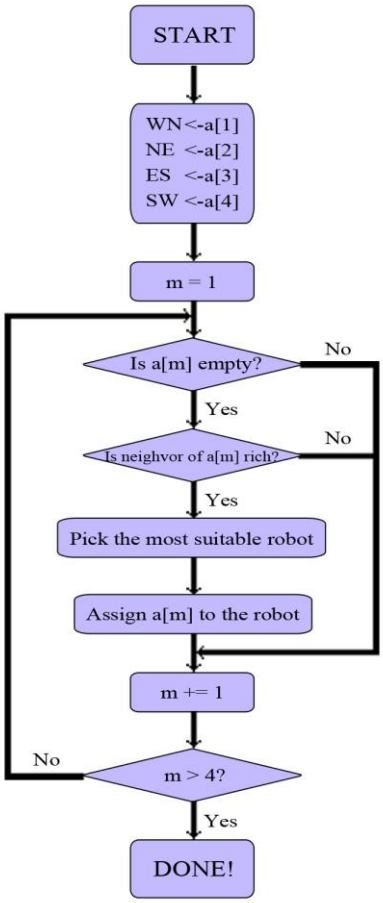


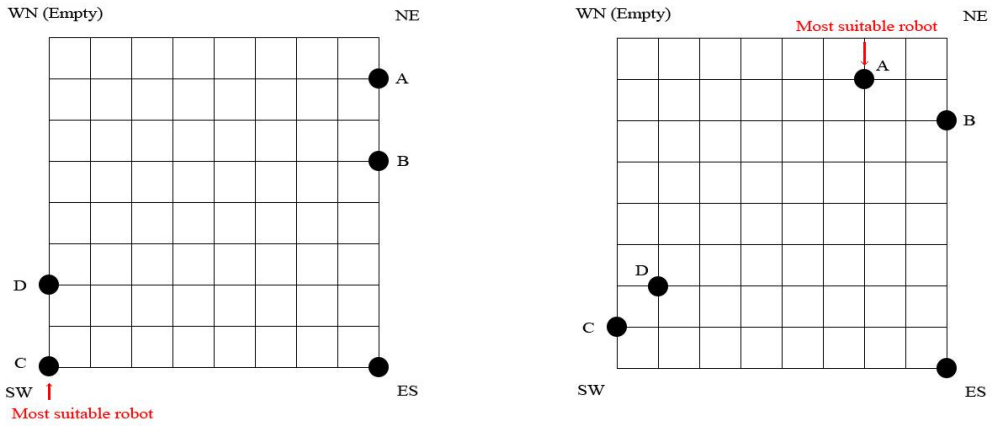
Figure 3.8 Reallocation Algorithm

Now we will give a brief description of picking algorithm here. As the name implies, picking algorithm is proposed to pick the most suitable robot which will be assigned to a new strategy. The algorithm can be divided into two separate rounds:

1. If the empty strategy is *WN* or *ES*, the most suitable robot will be among the robots which follow the strategy *NE* or *SW*. In the first round, the algorithm will choose the one which is closest to the northern or southern boundary as the most suitable robot. For example shown in Figure 3.9(a), empty strategy is strategy *WN*. There are two robots (A and B) following the strategy *NE*, and another two robots (C and D) following the strategy *SW*. The distance from robot A and B to the northern boundary are one and three respectively, whereas those from C and D to the southern boundary are zero and two, respectively. Thus the algorithm will pick the robot C as the most suitable robot since it holds the shortest distance to the southern boundary.

If there is more than one robot holding the closest distance to the northern or southern boundaries, the algorithm will start the second round, which aims to choose the robot which is farthest to its closer easternmost or western boundary. For example, in Figure 3.9(b), strategy *WN* is an empty strategy, and robot A and B following the strategy *NE*, and robot C and D following the strategy *SW*. The shortest distance from robot A to the northern boundary is the same as that from B to the southern boundary. So we have to check the distance from the robots to their closer eastern or western boundary. We can see that robot A is closer to the eastern boundary while C is closer to the western boundary. The algorithm will choose the most suitable robot according to the distances of A-to-East and C-to-West. Thus A is chosen as the most suitable robot to be assigned to strategy *WN*.

If several robots hold the equal shortest distance in the first round selection and longest distance in the second round selection, the algorithm will pick one among them randomly.



(a) (b)

Figure 3.9 Picking Algorithm

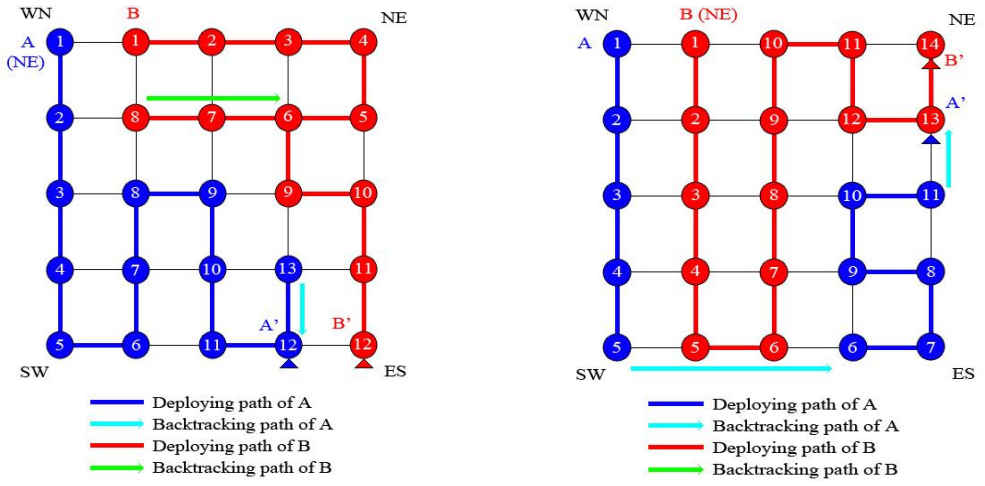
2. When the empty strategy is *NE* or *SW*, the first round of method is similar to the one in the situation above. The algorithm will choose the robot which is closest to the easternmost or western boundary as the most suitable robot.

If there is more than one robot holding the closest distance to the easternmost or western boundaries, the algorithm will select the one which is farthest to its closer northernmost or southern boundary in the second round.

The reason of proposing picking algorithm in this way is to make sure each subarea is independent and not cut into pieces by other subareas. Figure 3.10 and Figure 3.11 give us an illustration.

Figure 3.10(a) and (b) is a comparison of first round selection. There are two robots (A and B) in the figures, both of which are assigned to strategy *WN* at the first beginning. The algorithm will search for the empty strategy following the order *WN*> *NE*> *ES*> *SW*. After the first empty strategy *NE* gets found, the algorithm will check if its neighbor strategies are rich. The capacity of *WN* is two, thus one of robots A and B will be selected as the most suitable robot to be assigned new strategy.

In Figure 3.10(a), A is assigned with the new strategy *NE* while B follows the original one-*WN*, which meets the design of first round selection of picking algorithm. The sensors placed by each robot form a whole and independent subarea, which is not cut into pieces. However, Figure 3.10(b) shows a different example, which provides an inverse strategy allocation. We can see robot B keeps moving towards south, A has to move to the same direction since it is restricted by B and cannot move to east. After they reach the southern edge, B moves to northeast whereas A has to backtrack to southeast area to deploy sensors there. As a result, the subarea of robot A is cut into two separate pieces by subarea of robot B. Therefore, the principle of the first round selection in picking algorithm is proved reasonable.



(a) (b)

Figure 3.10 First Round of Picking Algorithm

Figure 3.11 explains the advantage of the second round selection. In Figure 3.11(a), two robots are assigned to different strategies according to the second round selection of picking algorithm. The subarea of each robot is independent. In Figure 3.11(b), moving strategies are allocated oppositely, which causes the same problem as shown in Figure 3.10(b). The subarea of A is divided into two segments. The second round of picking algorithm is proved workable with this example.

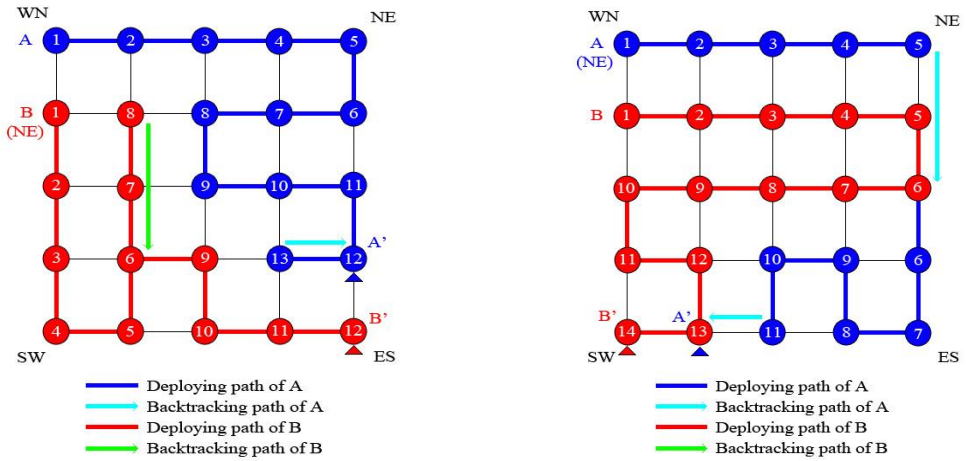


Figure 3.11 Second Round of Picking Algorithm

Combining reallocating algorithm with picking algorithm, a brief description of how strategy reallocation works is given, with Figure 3.6 as an example:

1. Searching for empty strategies following the order *WN*> *NE*> *ES*> *SW*. Then the strategy *ES* is found as the first empty strategy.

2. Seeking rich strategies among the neighbor strategies of *ES*, which are *NE* and *SW*. Then strategy *SW* is found as a rich strategy, so one of the robots following strategy *SW* will be assigned to the new strategy *ES*.

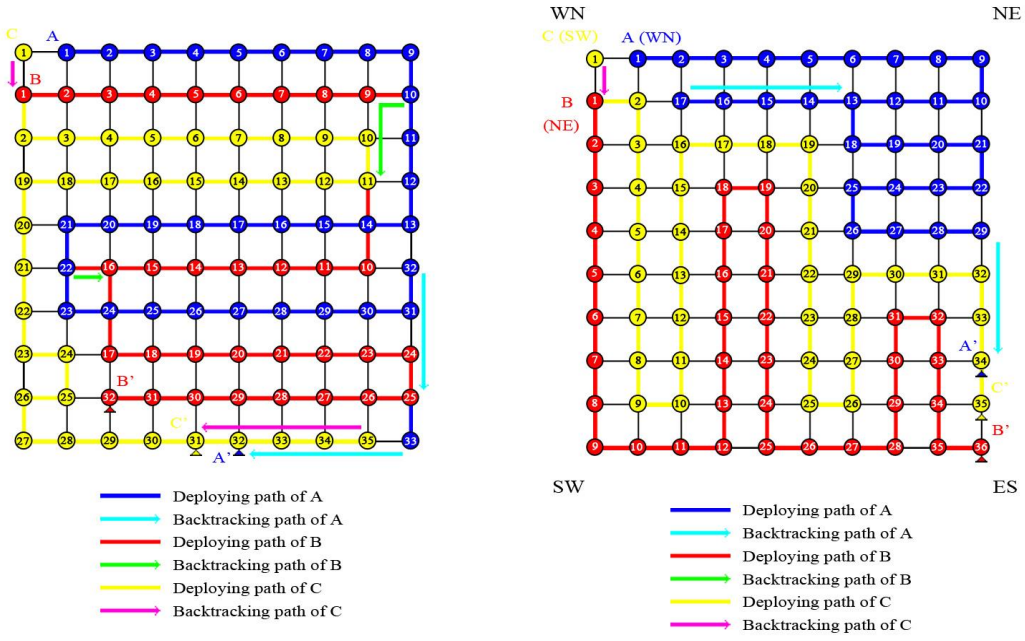
3. Selecting the most suitable robot according to first round selection in picking algorithm. In this example, empty strategy is *ES*, so we should find the robot which is closest to the northern or southern boundary. There are two robots on the southern boundary. The distance from them to the southern edge is 0, which is the shortest distance. Because there are more than one of robots holding the shortest distance, we have to the second round of selection.

4. In the second round of selection, the one which is farthest to its closer easternmost or western boundary will be chosen as the most suitable robot. The robot on the right meets the requirement, so it will be chosen as the robot which will be assigned with strategy *ES*.

5. After reassigning strategy *ES* to the most suitable robot, the capacity of *WN*, *NE*, *ES* and *SW* moving strategies are two, one, one and two respectively. No empty strategy can be found anymore, so the reallocation will stop.

Figure 3.12 and Figure 3.13 show the fully covered ROI in Figure 3.5 and Figure 3.7. In (a), the robots follow their original strategies, whereas robots are assigned with new strategies according to the reallocating and picking algorithm in (b).

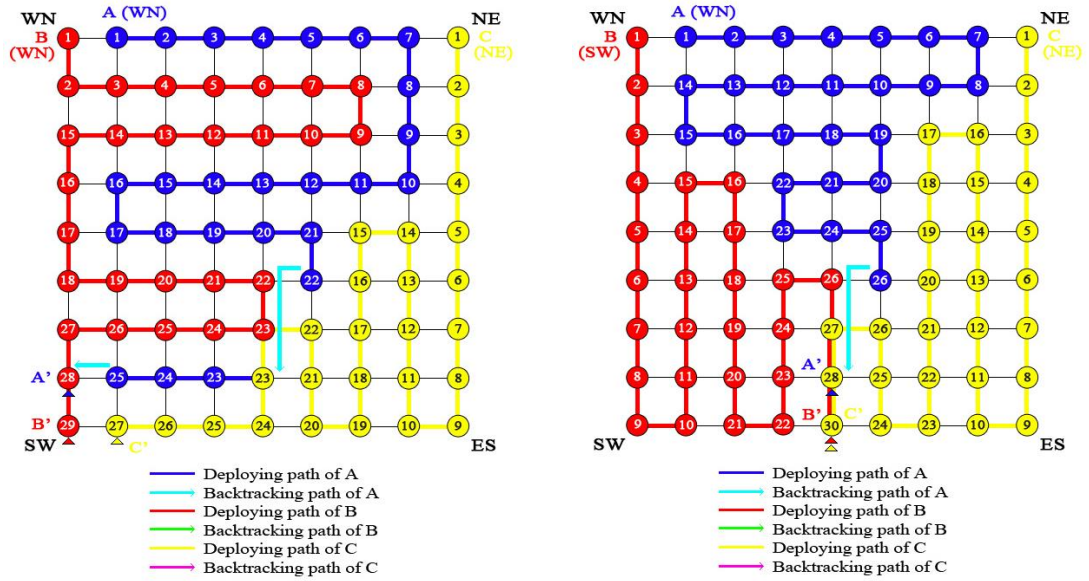
In Figure 3.12(a), all the robots follow strategy *WN*. On the contrary, in Figure 3.12(b), robot A follows the original strategy *WN*, while robot B and C are assigned to new strategies *NE* and *SW* respectively. We can see all the three subareas are cut into piece by each other in the first figure. However, in the second one, sensors deployed by robot A form an independent area, but subareas of robot B and C merge into each other. This problem will be solved later with combination algorithm.



(a) Before reallocation (b) After reallocation

Figure 3.12 Fully Covered ROI of Figure 3.5

Figure 3.13 shows the fully covered ROI in Figure 3.7. In the first figure, the subareas of robot A and B merge into each other since these two robots follow the same moving strategy. While robot B is assigned with strategy *SW* after reallocation in the second picture, which makes the areas more regular and independent.



(a) Before reallocation (b) After reallocation

Figure 3.13 Fully Covered ROI of Figure 3.7

Therefore, the examples show the reallocation can indeed improve the sensor deployment algorithm, by making sensors placed by each robot form a more regular shape in order to make the coverage maintenance algorithm perform better.

However, sometimes subareas of different robots may merge into each other even after reallocation, like the areas of B and C in Figure 3.12(b). In this situation, a combination algorithm is required to shorten the moving distance before the beginning of the coverage maintenance algorithm.

3.1.2 Dead End Situation

EBD inherits sensor colors and dead end definitions from BTD. A robot in EBD will keep deploying sensors until it meets a dead end. In a dead end situation, it does not backtrack like the robot does in BTD algorithm. Instead, it will search for the nearest white sensor to resume deploying.

The principle of searching for a white sensor is proposed by expanding the ring principle, used in searching for destination in routing. It is proposed as follows:

Assuming there is a robot gets stuck at sensor 0, it will send searching messages in four geographic directions for a white sensor. If sensor 0 does not have an adjacent obstacle neighbor, the messages will be received by four sensors, which are the four one hop neighbors of sensor 0. If there is at least one white sensor among them, the all the white sensors will send messages back. Once the robot receives a message sent back, it will consider the white sensor as the nearest white sensor and move to it. If there are more than one of messages received at the same time, the robot will pick one randomly.

If there is no white sensor among the one hop neighbors, the one hop neighbors will send messages in their four directions, and the messages will meet eight sensors, which are two hop neighbors of sensor 0. The searching messages will spread in this way until a white sensor is found.

Figure 3.14 shows the one hop neighbors to four hop neighbors of sensor 0 in an environment with an obstacle.

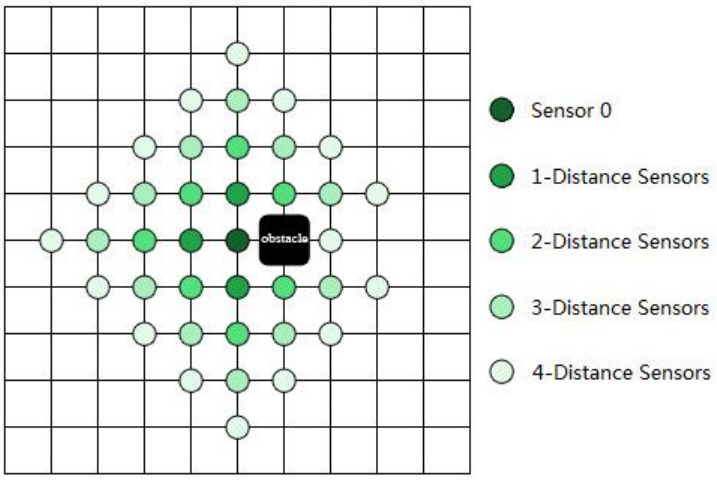


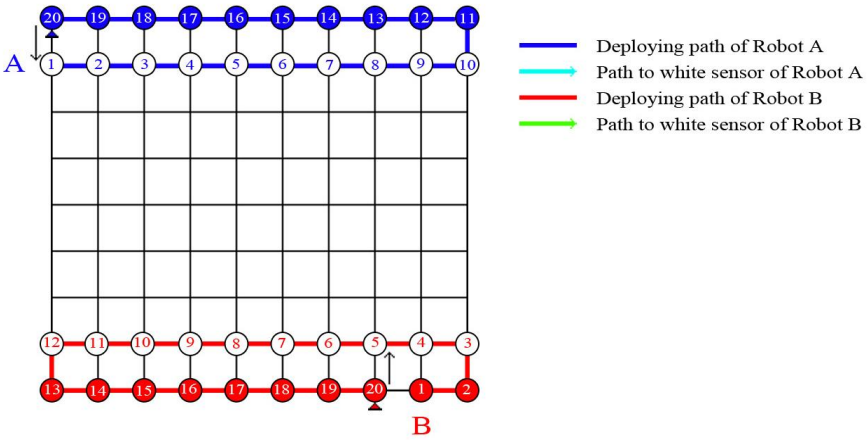
Figure 3.14 Neighbors of Sensor 0

Figure 3.15 gives an example of backtracking in EBD. In this example, robot A is assigned to strategy *WN* and robot B follows the *ES*. Robot A places sensors on the top two rows before it meets a dead end at the northwest corner. In Figure 3.15(a), robot A chooses to move to the nearest white sensor 1, instead of the back pointer sensor 10, which decreases unnecessary movements. Figure 3.15(b) shows the fully

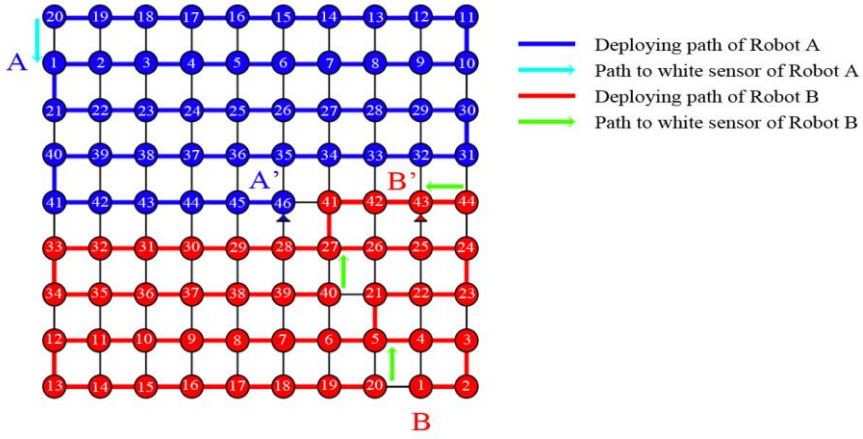
covered ROI, where we can see that robot A places 46 sensors while robot B places 44 sensors, the working load of two robots has been balanced, and the length of backtracking path is only 4.

Compared with the backtracking in EBD, Figure 3.16 shows the routes and backtracking path of robots following BTD approach in the same environment. We can see that both robots have long backtracking path. In Figure 3.16(a), robot A backtracks from sensor 20 to 10. We can observe from Figure 3.16(b) that the two robots deploy 41 and 49 sensors respectively, and the total distance of backtracking path is 19, which is much longer than that in the last figure.

Thus, backtracking to the nearest white sensor can shorten the backtracking path and balance the workload.

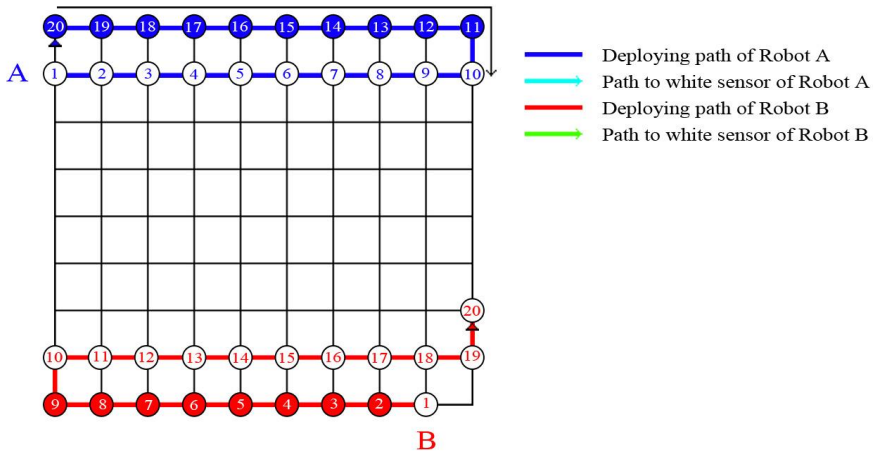


(a) Backtracking to the Nearest White Sensor

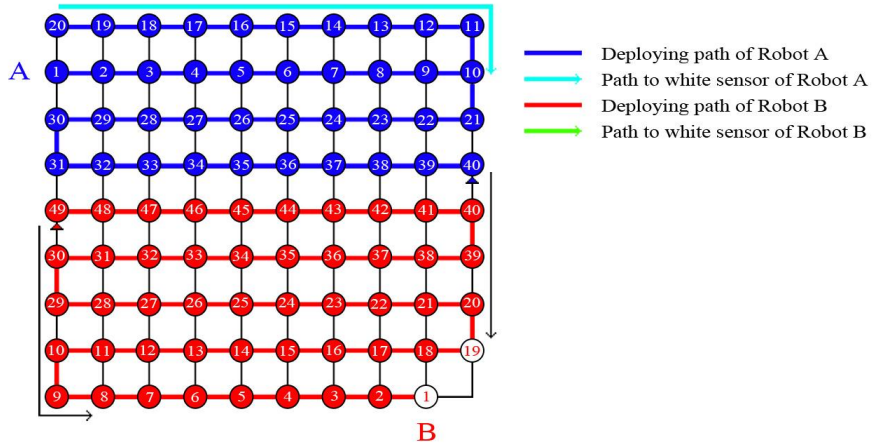


(b) Full Coverage

Figure 3.15 Backtracking in EBD



(a) Backtracking to the Back Pointer



(b) Full Coverage

Figure 3.16 Backtracking in BTD

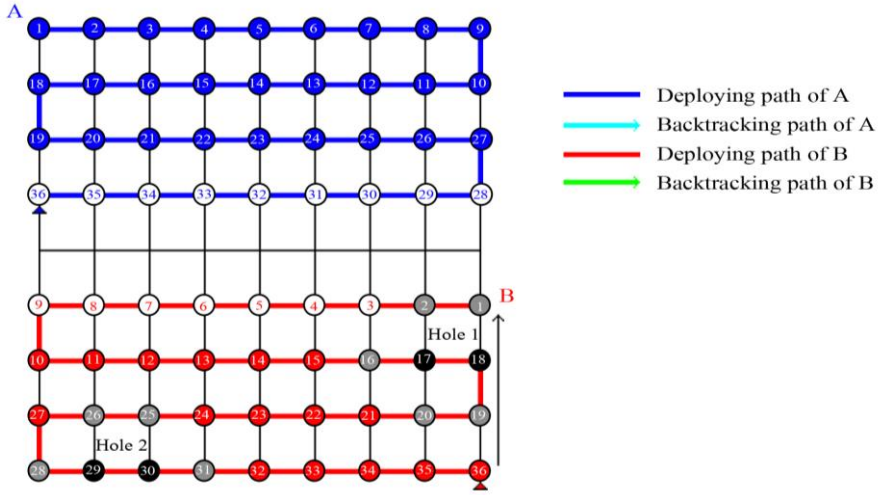
3.1.3 Gray Sensor Situation

In a failure-prone environment, it is possible for sensors to fail after long-term of usage. Failures generate sensing holes in the ROI, which affect the accuracy of recorded data. As we mentioned in Section 2.2.3 when we introduced BTD approach, a sensor will color itself "*gray*" if at least one of its one hop neighbors is marked failed. Therefore, all sensing holes are supposed to be surrounded by gray sensors.

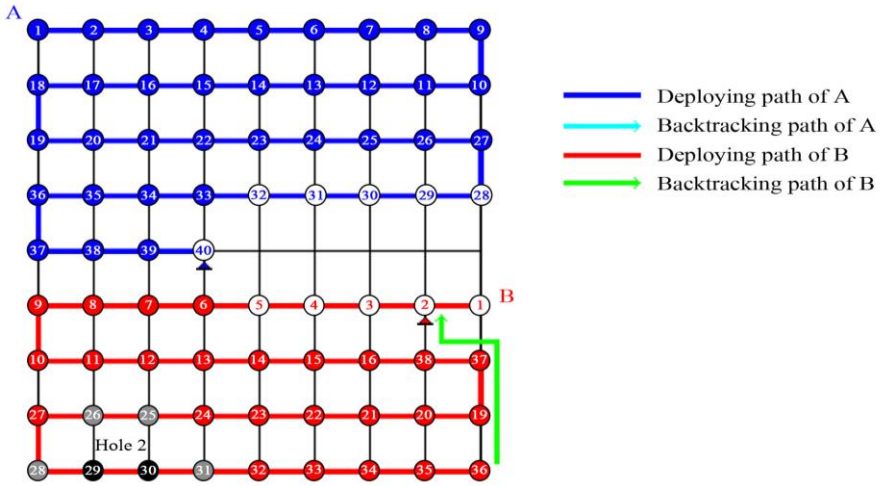
If the route of a robot is interrupted by a gray sensor, or a gray sensor is found in white sensor searching, the robot will change its task from deploying sensors to filling sensing hole first. After the hole is patched, the robot will move to its original destination and resume deploying.

Figure 3.17 gives an example of gray sensor situation. Robot B gets stuck at the southeast corner, then it sends searching messages for a white sensor; however, the messages encounter a gray sensor, which is sensor 19, and return with the position of it, as shown in Figure 3.17(a). Then robot B decides to patch hole 1 firstly. After it replaces sensor 17 and 18 continuously, all the gray sensors on the boundary of sensing hole 1 change to black or white. In Figure 3.17(b), the robot moves to the nearest white sensor 2 to resume deploying. The two robots keep placing sensors until the ROI gets fully covered, shown in Figure 3.17(c).

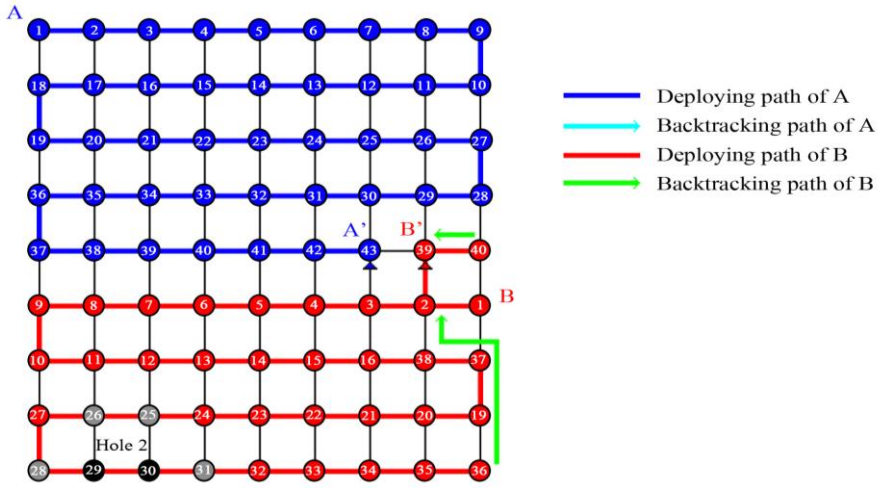
In Figure 3.17(a), sensing hole 2 is not on the backtracking path of any robot. So it will be left untreated until the coverage maintenance phase starts. When the robots detect there is no white sensor in the ROI, the coverage maintenance algorithm will start, and hole 2 will be patched then.



(a) Getting Stuck



(b) Hole Covering



(c) Full Coverage

Figure 3.17 Sensing Hole Encounter

3.1.4 Termination of Sensor Deployment Phase

The first phase of algorithm will terminate when there is no white sensor in the whole area. No white sensor here means that every single grid point is covered with a sensor, and there is no empty point in the ROI. However, sensors may break down after they keep working for a while, which may cause sensing holes in the area and decrease the accuracy of measurement. Then the robots will move to the second phase of the protocol by waiting for failure reports in the ROI.

3.2 Coverage Maintenance

Once the sensor deployment algorithm finishes, the ROI is supposed to be fully covered by sensors. Although sensors are designed with low energy consumption, they cannot work permanently without breakdown. Sensor failures leave sensing holes in the ROI, which will cause inaccuracy of the information gathered by the network. In BTD algorithm, sensing holes can only be detected when they affect backtracking of robots. Thus the BTD algorithm terminates with holes untreated in the ROI, which cannot guarantee full coverage in failure-prone environment.

In this section, EBD algorithm will be extended in order to maintain the network to work normally by detecting and patching sensing holes in failure-prone environment.

3.2.1 Subarea Combination

After the ROI is fully covered, robots will move to the central points in their subareas waiting for sensing holes. Once a sensor failure occurs, a robot will move to this sensor and replace it with a new one which works normally. Moving to a central point may increase the moving distance but it will save hole patching time.

Failures occur randomly in the whole ROI since we assume that sensors are homogeneous. The probability of failing for each robot is equal. Thus the average moving distance of a robot is in proportion to how regular the area is. The more regular the subarea is, the shorter distance the robot moves. This is the reason why we try to improve the deployment algorithm to let the sub-areas more regular.

Sometimes, the shape of subareas may not be as regular as we expect; for example, the areas of robot B and C in Figure 3.12(b), and the areas of robot A and B in Figure 3.13(a). Irregular subareas will result in a longer moving distance in coverage maintenance algorithm. Therefore, we have to combine the subareas which merge into each other to make the combined area more regular.

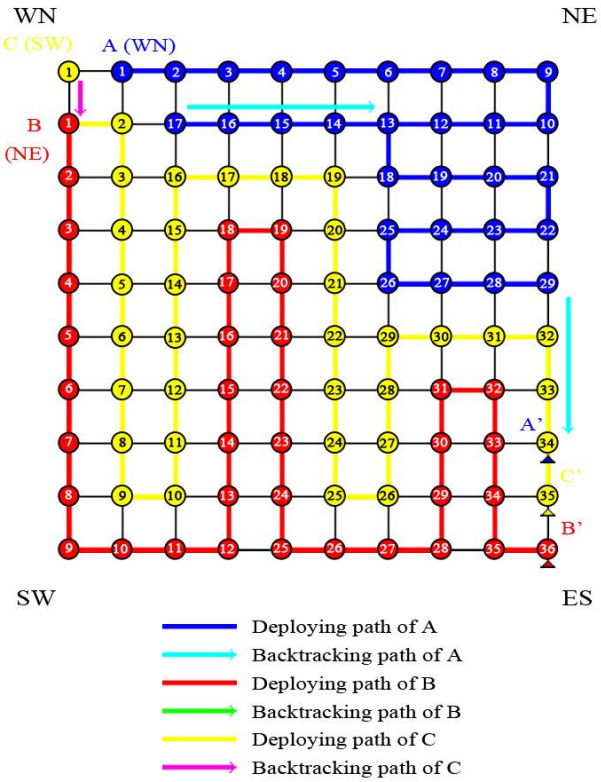
"*Combine*" means to treat several subareas as a large whole region. For example, combining subareas of robot A and B algorithmically means that A and B will share the information of the position of all the sensors they deploy. Also, the two subareas are treated as a whole area and both A and B are responsible for the new area. If there is a sensing hole in any one of these two areas, it will be considered as being in the large area and both A and B have the responsibility to fix it.

After combining several subareas into one, all robots responsible for these areas should be responsible for the large new area. They will choose new central points which minimize the sum of average moving distances. After moving to the central points, robots will wait there until the failure occurs.

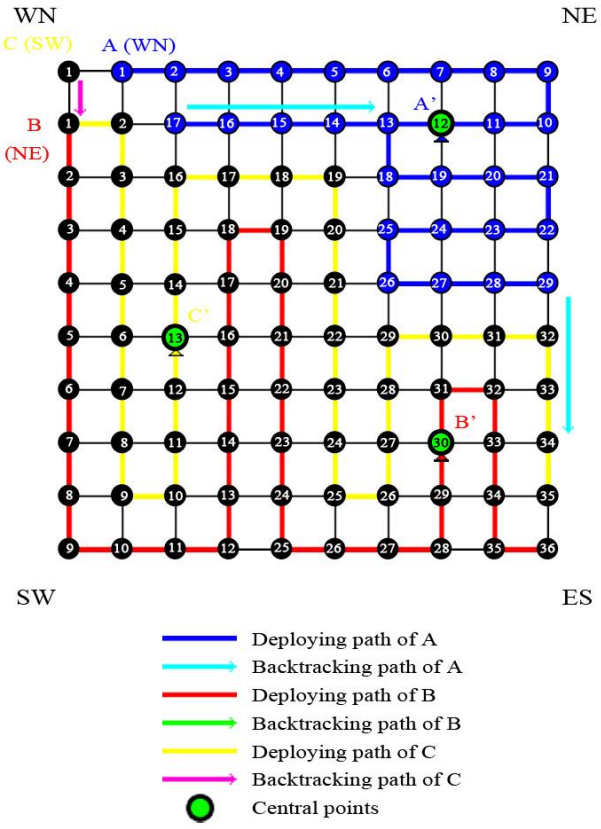
The subareas of robots which are close to the same corner and are not assigned with neighbor strategies after reallocation should be combined. For example, in Figure 3.18, both of robot B and C are close to the northwest corner. After reallocation, they are assigned with *NE* and WS respectively, which are not neighbor strategies. So, the subareas of these two robots should be combined together. In the coverage maintenance algorithm, we will consider the two subareas as one.

The combination principle is only related to the location of starting points and the strategies that robots follow. Thus, which areas should be combined after deployment is already known before the sensor deployment phase starts.

Take Figure 3.18(a) as an example again. After each point in the ROI is placed with a sensor, robot B and C will combine their subareas by sharing the position and sequential number of their sensors. After combination, these two areas will be considered as a large whole area, which is the black region in (b).



(a) Fully Covered ROI

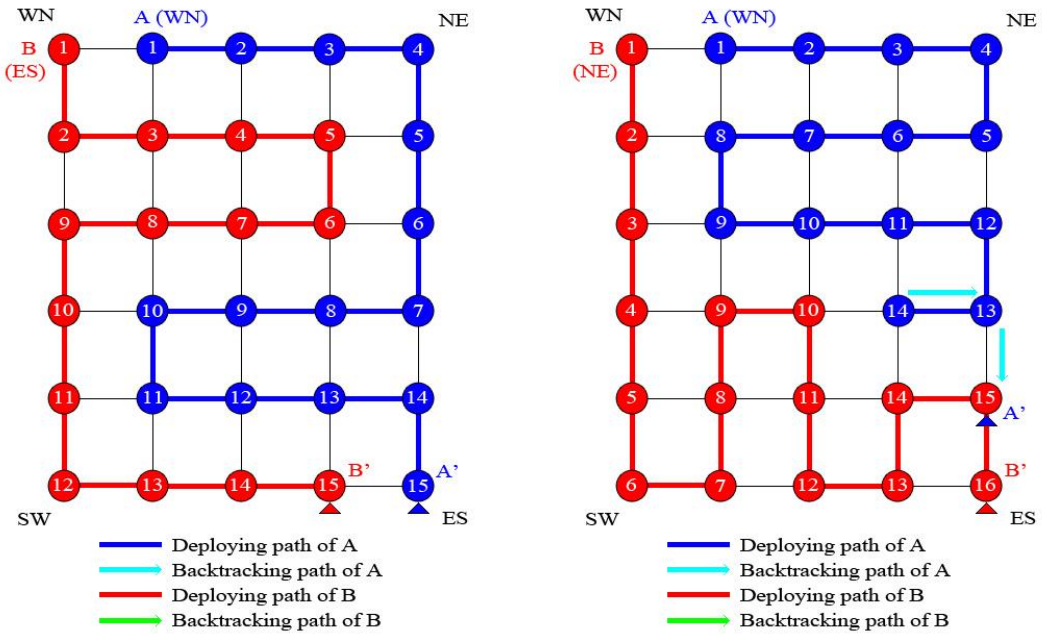


(b) the ROI after 'Combination'

Figure 3.18 Combination

The reason of proposing combination algorithm in this way is that subareas merge into each other as a whole when and only when they meet the conditions above. The algorithm combines these subareas into a larger and more regular piece. Take Figure 3.19(a) as an example, the two robots follow *WN* and *ES* respectively, which is not a pair of neighbor strategies, so the subareas merge into each other.

If two robots follow a pair of neighbor strategies respectively, that means that one moves horizontally whereas the other one moves vertically. As a result, the sensors they deploy will gather around two diagonal corners, and the boundary of these two areas will be easy to tell. Figure 3.19(b) gives us an example, in this instance, A and B follow a pair of neighbor strategies which are *WN* and *NE*. We can see both of the two subareas are in good shape, and we do not need to combine them.



(a) Allocated with Diagonal Strategies (b) Allocated with Neighbor Strategies

Figure 3.19 How Combination Benefits Subarea Shapes

3.2.2 Choosing Central Points

In this section, we will introduce after some robots combining their areas with each other, if there is no sensing hole to patch or the robots just finished covering a hole, they will be free and waiting for the next sensing hole coming out. During this period of time, robots will move to the central points in their subareas. The central point can

be considered as the point holds the minimum sum of distance between itself and all other points in the subarea. Since the sensing hole appears randomly in the whole ROI, waiting at the center points can minimize the average moving distance in probability.

The robot which does not share information with others simply picks a central point in its own subareas. While for those robots which share information with others, they pick a set of central points, of which number is equal to the amount of robots in the combined large area. These central points minimize the sum of average moving distance.

Take Figure 3.18(b) as an example, robot B and C combine their subareas into one, so they pick two central points in the new area, which are B' and C'. Whereas robot A does not share its sensors with any robot, it selects a central point A' in its own subarea.

The approach used to find the position of a central point in an area is proposed as follows:

1. Assuming that the number of sensors deployed in this area is *n*, the coordinates of these sensors are: $E = \{(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)\}$.

2. Finding (X, Y) in assemblage *E* which minimizes the average distance *D*,

$$D = \frac{1}{n}\left[\sum_{k=1}^{n}\left(|X - x_k| + |Y - y_k|\right)\right],\ (X, Y) \in E \tag{3.1}$$

Thus, (X, Y) is the coordinate of central point we are looking for, and the robot will move to this spot waiting for sensing holes when it is free.

If there are more than one coordinates meet the request, the robot will move to the one which is closest to its current position.

3. Every time when a robot finishes fixing a sensing hole, it will replace the coordinate of failed sensors stored in its memory with the new ones. Thus, the coordinate of central point will update since the boundary of subarea and the total number of sensors may change if the hole is just on the edge of several subareas.

The method of picking several central points for robots sharing a large area is similar to the approach above. Assuming that the number of robots is *m*, the coordinates of central points is a set of (X,Y) which has *m* elements. It can be defined as: $R=\{(X_1,Y_1),(X_2,Y_2)\ldots(X_m,Y_m)\}$, which should meets the request below,

$$D=\frac{1}{n}\left[\sum_{k=1}^{n}\left(|X_{\min}-x_k|+|Y_{\min}-y_k|\right)\right],\ (\mathrm{X}_{\min},\mathrm{Y}_{\min})\in R \tag{3.2}$$

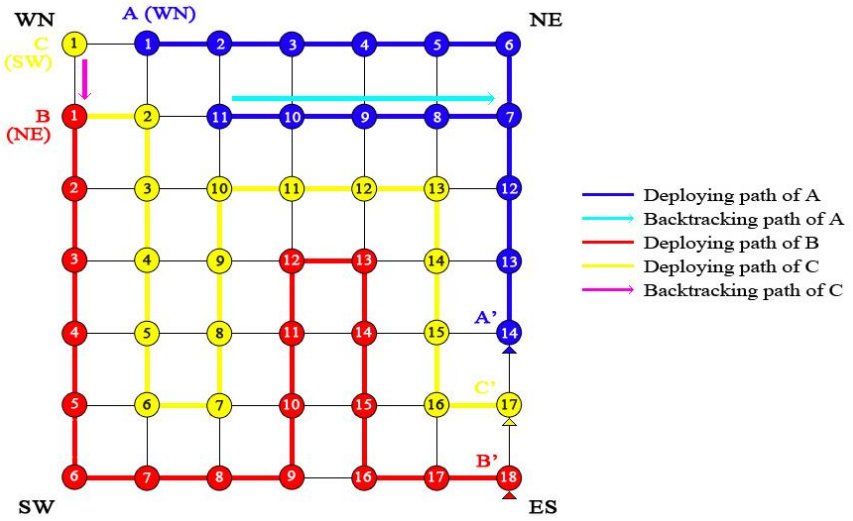
$(X_{\min},Y_{\min})$ in Equation (3.2) is not a constant coordinate, $\forall(x_k,y_k)\in E$, $(X_{\min},Y_{\min})$ is the coordinate which minimizes *d* in Equation (3.3). *d* represents the distance from (X,Y) to (x_k,y_k),

$$d=|X-x_k|+|Y-y_k| \tag{3.3}$$

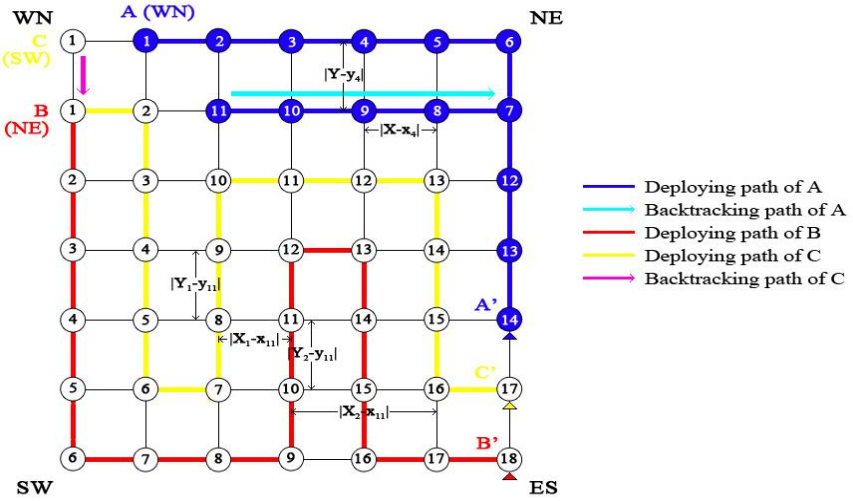
Take Figure 3.20 as an example. The fully covered ROI is illustrated in Figure 3.20(a), in which robot B and C should combine their areas into a new one. The ROI after combination is shown in Figure 3.20(b), the white area is the area combined by subareas of robot B and C.

If there is no hole to be patched, the three robots will find positions of central points and wait there for a hole coming out. In subarea of robot A, the distance between sensor 4 and 8 is $|X-x_4|+|Y-y_4|$, as shown in Figure 3.20(b), whereas, in the large white area, the distance between sensor 9, 16 to 11 are $|X_1-x_{11}|+|Y_1-y_{11}|$ and $|X_2-x_{11}|+|Y_2-y_{11}|$ respectively. It is obvious that the distance between sensor 9 and 11 is shorter than that between 16 and 11. Thus, we pick the smaller value, which is $|X_1-x_{11}|+|Y_1-y_{11}|$ as the distance from this set of coordinates to sensor 11. The $(X_{\min},Y_{\min})$ in this occasion is (X_1,Y_1), which is the coordinate of sensor 9.

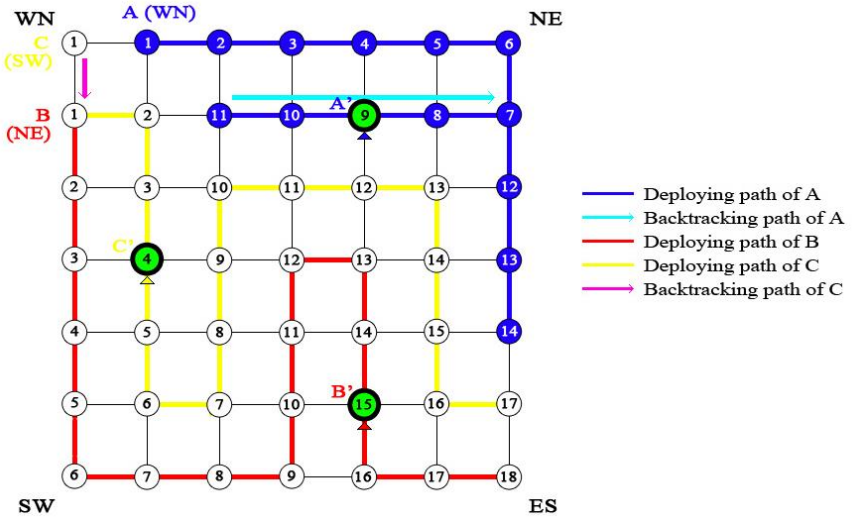
Finally, the central points are picked out and marked green with black boundary in Figure 3.20(c).



(a) Fully Covered ROI



(b) Distance Calculation



(c) Position of Central Points

Figure 3.20 Picking Central Points

If several coordinates meet the request at the same time, a random one will be selected. In this way, the coordinates of central points can be found.

Each robot will inform all sensors in its subarea with its current location once it moves to a new central point. A robot does so by sending messages to four geographical directions: west, east, north and south. The messages will spread in the subarea until the position of robot stored in all the sensors updated.

A robot which is selected to patch a sensing hole is called "*worker*" for this hole. The process used to find a worker is similar to the procedure of election. Several candidates which meet the requests are chosen before the worker is elected from them. The requests of selecting candidates and the standard of choosing the worker will be discussed in the following sections.

3.2.3 Candidates Selection

It is mentioned in Section 2.2.3 that a sensor colors itself gray when it is next to a failed sensor. Thus, sensing holes are surrounded by gray sensors.

Once a hole appears, a robot will be selected as the worker to patch it by replacing the failed sensors with new ones. Before the worker is chosen, a set of robots will be selected as candidates, which have the opportunity to patch this hole. One of the candidates will be elected to the worker. The request for a robot to be a candidate is given as follows:

If all gray sensors surrounding a hole are located in one subarea, no matter how many robots in this area, all of them have the opportunity to fix the hole. However, for sensing holes located on the boundary of several subareas, all the robots in these adjacent areas are candidates for the hole covering.

Figure 3.21 is an example extended from Figure 3.18. Blue sensors are deployed by robot A, while the large white area is combined by subareas of robot B and C. The central points are marked with black and green. It is assumed that there are three failures X, Y and Z. The candidate of worker for failure X is robot A; while the ones for failure Y are robot B and robot C. Since the gray sensors of failure Z are located in both two areas, all the three robots are candidates for hole Z.

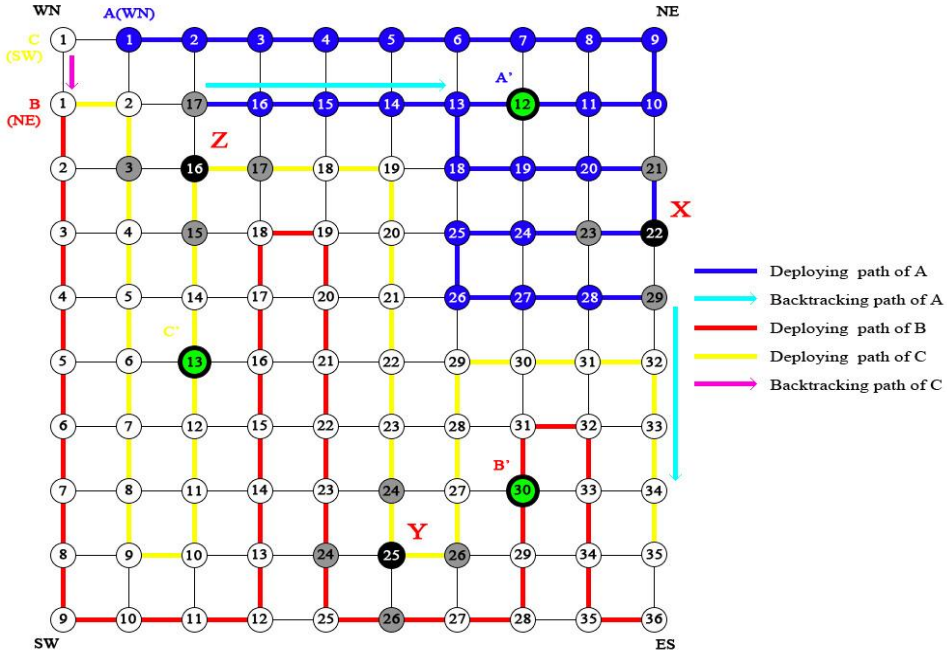


Figure 3.21 Candidates Selection

3.2.4 Worker Election

Once all the candidates are clear, the next step is to choose a worker among them, which can be considered as the most suitable robot to patch the hole. After the worker is selected, it will patch the sensing hole and return to the updated central point if there is no other task for it.

3.2.4.1 Free Candidates

If all the candidates are free, the one which is the closest to the sensing hole will be selected as the worker. The free here means having no hole to patch at present.

Take Figure 3.21 as example again. For hole X, the candidates is robot A since all gray sensors are located in the blue area. Thus, robot A will be selected as the worker of X. Whereas, for hole Y, robot B and C are candidates due to the hole is totally located in the large white areas. Comparing the distance from their location to the hole, B will be elected as the worker since it is closer to the hole. Hole Z is on the boundary of two areas. Robot C is the closest one to it, which will be chosen as the worker.

The gray sensors will send a failure report to the worker since they are aware of the location of it. The robot chosen as worker will move to the hole immediately once it receives the report.

If there is no obstacle in the ROI, the worker can move to the sensing hole directly, since the location of the failure and robot are known. Sometimes there are obstacles existing in the region which resist the movement of the worker. Several approaches can be used to help the worker find the shortest path from its current position to the sensing hole regardless of obstacles, such as A* algorithm [25].

It is notable that an approach can be used to simplify the movement complexity of the worker if the subarea is not a combined area, in which all sensors are deployed by the worker. This method guides the worker through an area with obstacle to the failure with the shortest moving distance as long as the worker is the only robot in this subarea.

The name of the approach is shortcut-to-hole. It is extended from shortcut method [24], which is used to search for back pointers in BTD algorithm. Since the sequence number of sensor where worker is situated and the closest gray sensor are known, the worker is able to find the shortest path following sequence numbers.

Assuming the sequence number of sensor where the worker located at is N_{worker}, the one of the closest gray sensor is $N_{gray-sensor}$.

If $N_{gray-sensor} = N_{worker}$, the failed sensor will be a one hop neighbor of the worker. The worker can move to it directly and patch the sensing hole.

If $N_{gray-sensor} > N_{worker}$, the worker will select the one hop neighbor with the sequence number which is the highest one among all the numbers lower than $N_{gray-sensor}$ as it next movement step, until it reaches its destination.

If $N_{gray-sensor} < N_{worker}$, the worker will choose the one hop neighbor with the sequence number which is the lowest one among all the numbers higher than $N_{gray-sensor}$ as it next movement step, until it reaches its destination.

Figure 3.22 illustrates how the principle guides the worker to reach its destination with the shortest moving distance and avoid obstacles. In Figure 3.22(a), robot A, B and C are situated at the central points marked with black and green when failure occurs. Robot C is chosen as the worker since it holds the shortest distance to the hole. After the failed sensor being removed, robot A and C will update their areas and central points, as shown in Figure 3.22(c).

Figure 3.22(d) gives another example of how a robot bypasses an obstacle following shortcut-to-hole approach. In this figure, it is clear that robot B passes by the obstacle and reaches to sensor 22 from sensor 12.

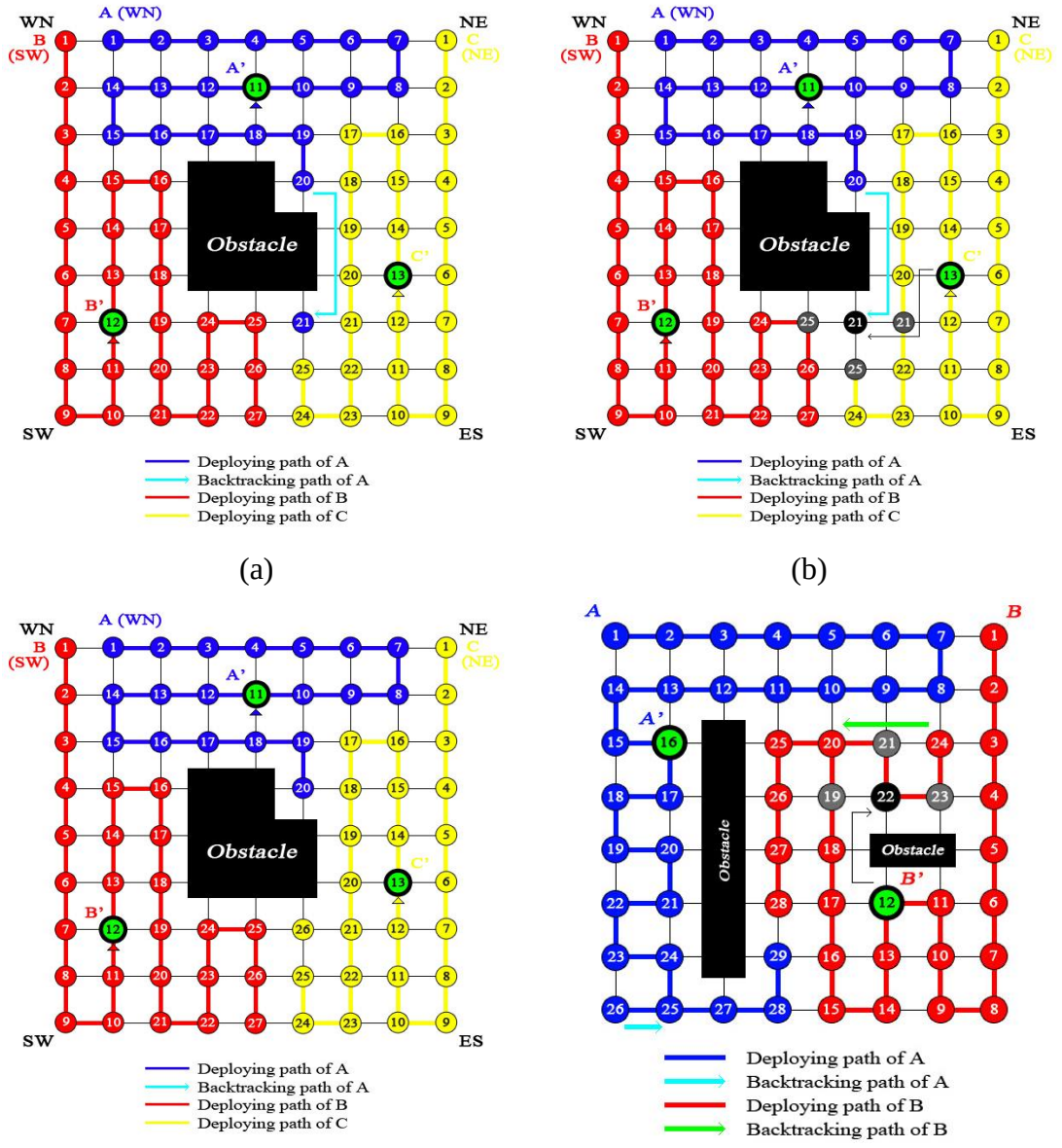


Figure 3.22 Shortcut-to-Hole Approach

3.2.4.2 Busy Candidates

Those candidates which are busy patching a hole should be taken into consideration as well in the worker selection. The way to calculate the distance from their current position to the hole is different from what has been discussed in the last section.

If there is only one candidate for a hole patching task, although it is currently busy patching another hole, it will be chosen as the worker. The worker will move to the next hole once it finishes the one it is working on.

If there are several candidates for one task, the one holds the shortest distance to the hole will be selected as the worker, which is the same as that in free candidates scenario. The distance between the robots and the hole can be calculated directly if they are free. Whereas, for busy candidates, the distance consists of three independent parts, which are given as follows:

1. $D_{Candidate-Hole}$, the distance between current position of a candidate and the hole it is going to patch. $D_{Candiate-Hole}$ is particular for those candidates which have one task but are still on their way to the hole and have not started working yet.

2. Assuming the coordinates of a candidate and the hole are $(x_{candidate}, y_{candidate})$ and (x_{hole}, y_{hole}) respectively, we have

$$D_{Candidate-Hole} = |x_{candidate} - x_{hole}| + |y_{candidate} - y_{hole}| \tag{3.4}$$

However, for candidates which are not on their way but have already began patching a hole, $D_{Candidate-Hole} = 0$.

3. $D_{\mathrm{Re}main}$, the remaining size of the currently patched hole, which can be calculated with the coordinates stored in the gray sensors surrounding the failure.

Firstly, arranging all coordinates with the same *x* value in a descending order, which are listed as follows:

$$\begin{bmatrix} (x_1, y_{1,1}) & (x_2, y_{2,1}) & \cdots & (x_m, y_{m,1}) \\ (x_1, y_{1,2}) & (x_2, y_{2,2}) & \cdots & (x_m, y_{m,2}) \\ \vdots & \vdots & \ddots & \vdots \\ (x_1, y_{1,i}) & (x_2, y_{2,j}) & \cdots & (x_m, y_{m,k}) \end{bmatrix}$$

The remaining size of sensing hole can be calculated with the equation below:

$$\begin{aligned} D_{Remain} = & \left((y_{1,1} - y_{1,2} - 1) + (y_{1,2} - y_{1,3} - 1) + \cdots + (y_{1,i-1} - y_{1,i} - 1)\right) \\ & + \left((y_{2,1} - y_{2,2} - 1) + (y_{2,2} - y_{2,3} - 1) + \cdots + (y_{2,j-1} - y_{2,j} - 1)\right) \\ & + \cdots + \left((y_{m,1} - y_{m,2} - 1) + (y_{m,2} - y_{m,3} - 1) + \cdots + (y_{m,k-1} - y_{m,k} - 1)\right) \end{aligned} \tag{3.5}$$

The equation can be proved correct in the following way:

(1) Dividing the sensing hole into several columns with the sensors, in each of them, having the same *x* value.

(2) Calculating the size of sensing hole in each column.

The columns can be divided into two different types: boundary with no failed sensors and inside with failed sensors. Figure 3.23 indicates the two types of columns, the columns on line $x=1$ and $x=5$ are boundaries, whereas the ones on between them are inside.

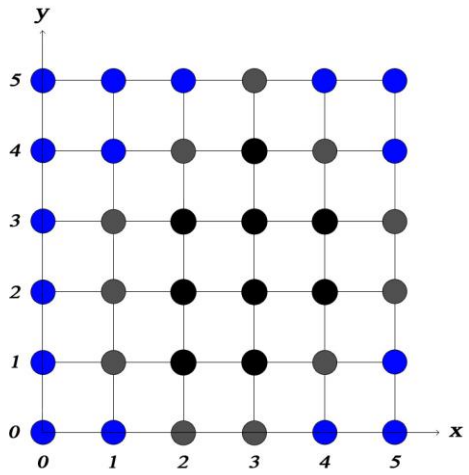


Figure 3.23 Columns of Sensing Hole

In an inside type column, there must be only two gray sensors in this column, which are on both sides of failed sensors, whereas, in a boundary type column, the number of gray sensors is not a certain value.

For the situation that only one gray sensor is in a column, it must be the boundary of a hole; then the size of hole in this column is 0.

If there are two gray sensors sharing a column, whose coordinates are (x, y_1) and (x, y_2), where $y_1 \geq y_2$. The size of sensing hole N_{size} between the two gray ones can be calculated with the equation below:

$$N_{size} = y_1 - y_2 - 1 \tag{3.6}$$

For boundary columns, the number of failed sensors is 0, which can be calculated with Equation (3.6) as well. Since after sequencing the coordinates in each column in a descending order, each *y* value is larger than the next one by 1. Thus, Equation (3.6) can be extended in the following way to fit the boundary scenario:

$$N_{size} = (y_1 - y_2 - 1) + (y_2 - y_3 - 1) + \ldots + (y_{k-1} - y_k - 1) \tag{3.7}$$

where $(y_1, y_2, \ldots, y_k)$ are the set of *y* values in one column sequenced in a descending order. With the element in the each bracket equals to 0, the value of the whole equation, which is N_{size}, equals to 0.

(3) In this way, the whole size of sensing hole can be calculated with the number of failed sensors in each column are known.

Take Figure 3.23 as an example of hole size calculation. Firstly, sequencing the coordinates of gray sensors and listing them the following matrix:

$$\begin{bmatrix} (1,3) & (2,4) & (3,5) & (4,4) & (5,3) \\ (1,2) & (2,0) & (3,0) & (4,1) & (5,2) \\ (1,1) & & & & \end{bmatrix}$$

The remaining size D_{Remain} can be calculated with the equation below:

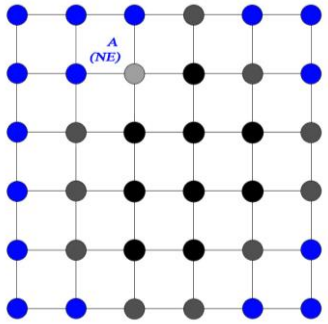
$$\begin{aligned} D_{Remain} &= \left((y_{1,1} - y_{1,2} - 1) + (y_{1,2} - y_{1,3} - 1) + \cdots + (y_{1,i-1} - y_{1,i} - 1)\right) \\ &\quad + \left((y_{2,1} - y_{2,2} - 1) + (y_{2,2} - y_{2,3} - 1) + \cdots + (y_{2,j-1} - y_{2,j} - 1)\right) \\ &\quad + \cdots + \left((y_{m,1} - y_{m,2} - 1) + (y_{m,2} - y_{m,3} - 1) + \cdots + (y_{m,k-1} - y_{m,k} - 1)\right) \\ &= (y_{1,1} - y_{1,2} - 1) + (y_{1,2} - y_{1,3} - 1) + (y_{2,1} - y_{2,2} - 1) + (y_{3,1} - y_{3,2} - 1) + (y_{4,1} - y_{4,2} - 1) \\ &\quad + (y_{5,1} - y_{5,2} - 1) \\ &= (3\text{-}2\text{-}1) + (2\text{-}1\text{-}1) + (4\text{-}0\text{-}1) + (5\text{-}0\text{-}1) + (4\text{-}1\text{-}1) + (3\text{-}2\text{-}1) \\ &= 0 + 0 + 3 + 4 + 2 \\ &= 9 \end{aligned}$$

4. $D_{Hole-Hole}$, the distance from the currently patched hole to the next one.

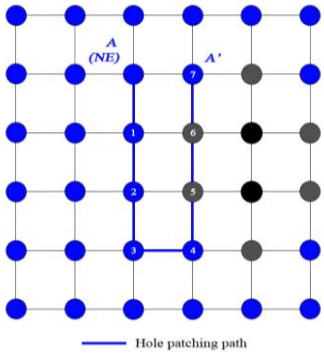
The position of the next hole is known since the coordinate of a gray sensor is carried by the failure report from the detector to all candidates. Thus, we need to find out which failed sensor will be remain last in the sensing hole, which can be seen as the starting point of the robot moving from the current hole to the next one.

In a sensing hole, the worker acts as it were in a single robot scenario. It moves in the hole following its strategy, and backtracks to the closest gray sensor when it gets stuck.

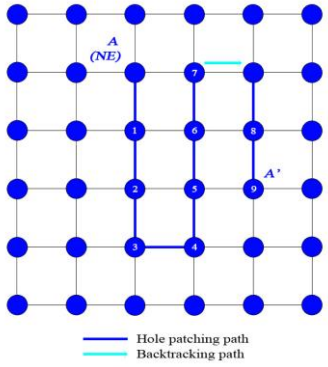
Figure 3.24 can be used as an illustration of how a worker moves when it is patching a sensing hole in the situation in Figure 3.23. The worker A starts from the light gray sensor in Figure 3.24(a) and moves towards south following the *NE* strategy. After replacing seven failed sensors, it gets stuck at sensor 7 in the second figure. Thus, it backtracks to the nearest gray sensor and keeps replacing sensors until the hole gets fully covered, shown in Figure 3.24(c).



(a)



(b)



(c)

Figure 3.24 Sensing Hole Patching

Therefore, the algorithm used in a single robot scenario can be used to find the last remained spot in the hole. With positions of this point and the gray sensor of the next hole, the distance can be obtained by Equation (3.8). Assuming the coordinate of the last spots is (x_{last}, y_{last}), while the gray sensor is on (x_{next}, y_{next}), we have:

$$D_{Hole-Hole} = |x_{last} - x_{next}| + |y_{last} - y_{next}| \tag{3.8}$$

With all values of $D_{Candidate-Hole}$, $D_{\mathrm{Re}main}$ and $D_{Hole-Hole}$, the distance from a busy candidate to the next sensing hole can be obtained by adding them together,

$$D = D_{Candidate-Hole} + D_{\mathrm{Re}main} + D_{Hole-Hole} \tag{3.9}$$

The D calculated with Equation (3.9) can be used in comparing distance between different candidates when choosing the worker. The candidate with the smallest D will win the opportunity to patch the hole.

It is notable that once a robot finishes covering a hole, it will move to the central point waiting for the next task. During its traveling period, it can be considered as free since it is able to move to the next hole immediately after receiving another task.

3.2.5 Termination of Hole Patching Phase

The hole patching algorithm does not have a specific terminating time. Robots will not stop working as long as they are workers of any hole in the ROI. When there is no failure report, these robots will turn to a sleep mode waiting on the central points, since no failed sensor needs to be replaced by it. Once receiving a failure report, they will turn to the work mode and start working on the hole patching. The switch between sleep and work enables robots to avoid wasting unnecessary energy and react quickly when a hole comes out.

If failure does not happen so often, half of the robots can be removed from the ROI, which enlarges the size of sub-area of each robot to twice the original size.

Chapter 4: Simulation Results

In this chapter, we evaluate the performance of the two phases (sensor deployment and coverage maintenance) of the proposed EBD through simulation. For the sensor deployment phase, we compare the performance of EBD with the existing BTD approach, whereas in the case of coverage maintenance phase, we use Dynamic Distributed Manager Algorithm (DDMA) with some slight modifications. Recall that in DDMA, one of the three algorithms proposed by Mei et al. [13] and described in Chapter 2, each robot serves simultaneously as the manager and hole fixer.

4.1 Sensor Deployment

In this phase, EBD will be compared with BTD approach in both failure-free and failure-prone environment. The reason of using BTD for comparison is that it only works in both single-robot and multi-robot scenario, whereas LRV and OFRD approaches are simply proposed to solve sensor deployment problems in single-robot scenario.

4.1.1 Simulation Environment

Our simulator is written in Java. During the simulation, we have assumed the sensing radius (r_S) and communication radius (r_C) of all the sensors to be the same, which are $r_S = \sqrt{2}$ and $r_C = 2r_S = 2\sqrt{2}$, respectively. We have considered a rectangular ROI with obstacles. Thus, the length of each virtual grid is $l = \sqrt{2}r_S = 2$ units. The environment is set up as shown in Figure 4.1, where the three black blocks represent obstacles. There are 192 grid points in the area with 176 of them are not occupied by obstacles, which can be deployed with sensors.

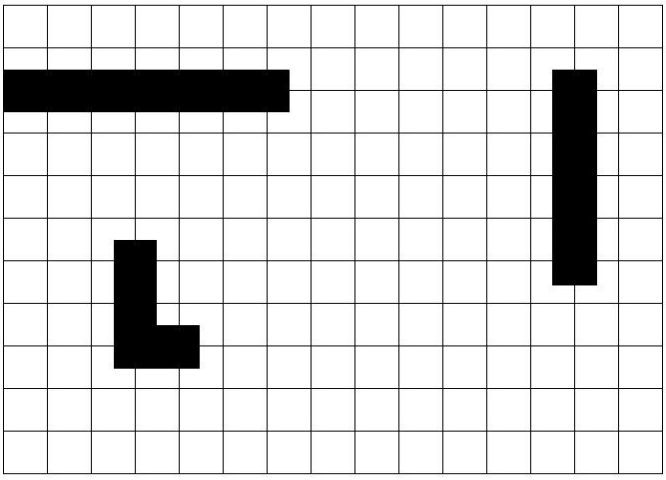


Figure 4.1 Simulation Environment

We placed *m* robots randomly in the ROI initially and each robot moves in a constant speed (i.e. 2 units in any four directions). We implemented both algorithms (BTD and EBD) with different number of robots, varying from 2 to 6. Robots drop a sensor as they move along grid points. The communication radius of the robots is same as that of the sensors.

For creating/simulating sensing holes of size *h*, we randomly pick a sensor as the first failed sensor; then, the next one is selected among its adjacent sensors which have not failed yet, until there are *h* sensors in the hole. During the simulation, we randomly generated 2 to 4 sensing holes in the ROI whose size is a random number varies from 1 to 6.

There are five metrics used to evaluate the performance; for each metric, we did simulation for twenty times and average the results. The five metrics are given below:

1. *Coverage Ratio (CR).* It is the ratio of grid points covered with sensors to the total number of points.
2. *Robot Moves (MV).* It is the number of moves of each robot during the deployment, and represents the time cost and the energy consumption for covering the ROI.
3. *Workload (WL).* It is the number of sensors a robot deploys. As part of the *WL*, we also report highest and lowest number of sensors that a robot deploys to illustrate how balanced it is.

4. *Robot Messages (RM)*. It is the number of messages sent by robots during the deployment which reflects the total energy consumption for communication among robots and sensors.
5. *Sensor Messages (SM)*. It is the number of messages sent among sensors during the deployment which reflects the total energy consumption for communication among sensors.

In reality, the *CR* should be maximized, whereas the other four metrics should be minimized for lower energy consumption and higher working efficiency.

4.1.2 Coverage Ratio

Firstly, we analyze the performance of the two algorithms (BTD and EBD) with respect to *CR*. It can be seen from Figure 4.2 that both algorithms can guarantee coverage higher than 90% percentage in a failure-prone scenario.

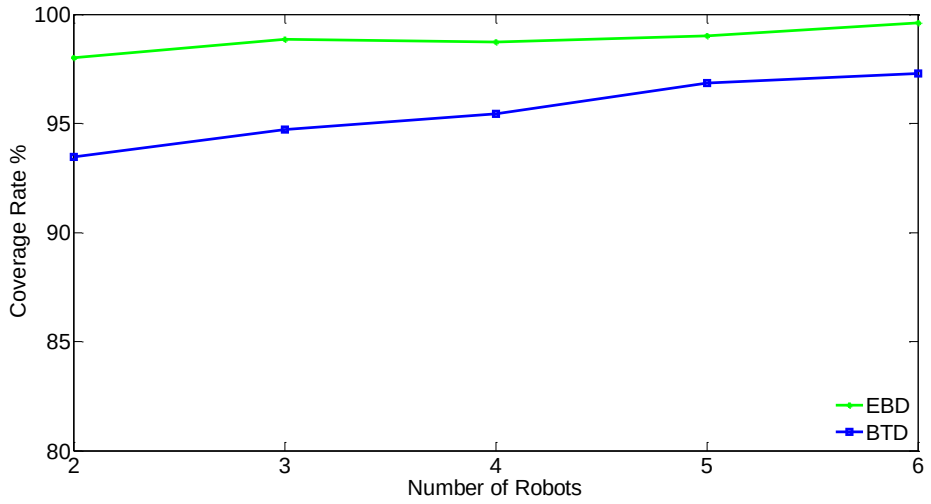


Figure 4.2 Coverage Ratio vs Number of Robots

In BTD, sensing holes that do not affect back tracking will be left untreated. In case of EBD, robots send searching messages for locating the nearest white or gray sensor when they get stuck, instead of backtracking to the back pointer. Thus, the probability of discovering a hole in the case of EBD is higher than that in the case of BTD. We can observe an ascending trend on both two algorithms, since the more robots we have in the ROI, the higher probability for them to discover the holes.

4.1.3 Robot Moves

The number of robot moves (*MV*) represents the energy and time consumption, which is shown in Figure 4.3.

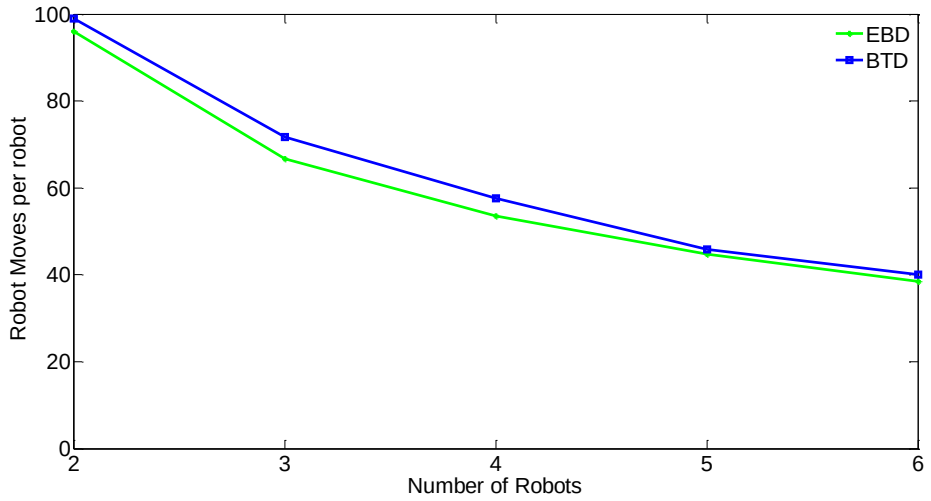
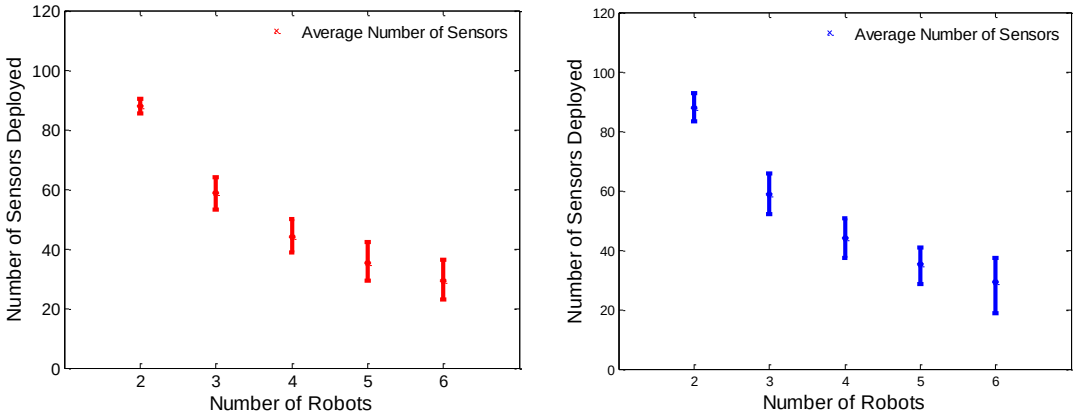


Figure 4.3 Robot Moves vs Number of Robots

In both cases, *MV* decreases as the number of robots increase, since the total movements are approximately the same, which equals to 200. *MV* in case of EBD is slightly lower (better) than that in the case of BTD, which indicates that EBD has lower energy and time consumption.

4.1.4 Workload

The workload of robots should be balanced to achieve a shorter deploying time. Figure 4.4 below shows robot workload with the maximum, minimum number of sensors deployed.



(a) Workload of EBD (b) Workload of BTD

Figure 4.4 Workload Comparison

Compared two figures in Figure 4.4, we can see that, although the workload appears same in both cases, the balance of workload is different. The workload is more balanced in the case of EBD than that in the case of BTD. The largest difference in workload in EBD is approximately 15, which occurs when $m = 6$, whereas the difference in workload in BTD varies between 10 to 25.

It can be noticed that the workload difference of EBD increases slightly when $m \geq 5$, since robots are more likely to get stuck when their number increases.

4.1.5 Robot Messages

Robot messages contribute to communication overhead for robots. In EBD, all the robot messages are sent in four geographical directions in search for the nearest white or gray sensor when the robots get stuck.

Whereas in BTD, robots send messages in two situations: back tracking and load balancing. In back tracking scenario, robots will send messages to erase back pointers stored in the sensors along forward path when they intend to move to the back pointer of another robot. Robots do so in case of the collision caused by another robot backtracking to the same back pointer. In load balancing scenario, robots send searching messages for back pointer stored in its one hop neighbors when it is stuck and cannot find back pointer along its backward path. If it still cannot find a back pointer, the robot will send searching messages to four geographical directions, and the messages will spread around the whole ROI for a back pointer.

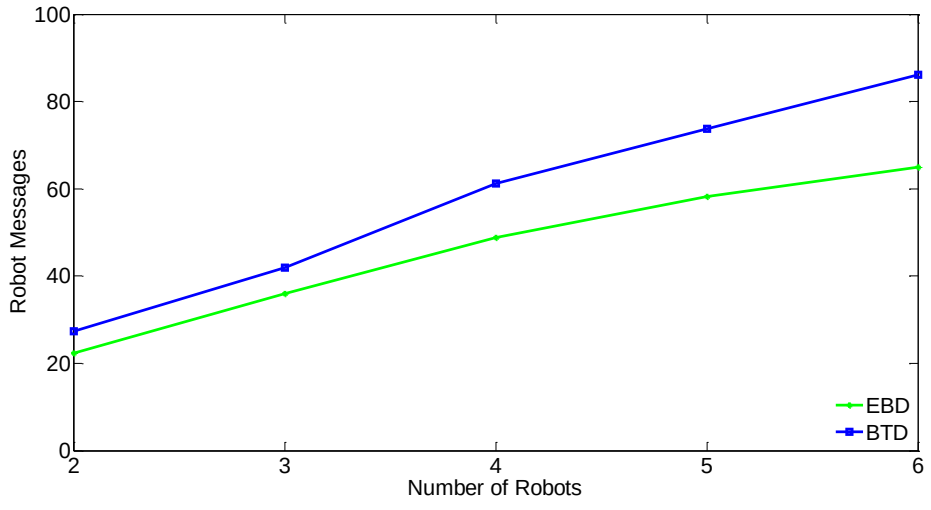


Figure 4.5 Robot Messages vs Number of Robots

Figure 4.5 shows the relationship between *RM* and *m*. We can notice that EBD outperforms BTD with respect to robot communication overhead. There is an ascending trend on both two lines. The ascending trend in case of EBD is due to the fact that robots are more likely to get stuck with more of them in the ROI since their possible moving directions may be occupied by other robots.

In the case of BTD, the number of robot messages increases in backtracking to back pointers of other robots or workload balancing, with the increase in number of robots for the same reason as that in EBD.

4.1.6 Sensor Messages

Sensor message is another type of communication overhead. Sensors transmit messages for a white or gray sensor in a dead end situation in EBD. While in BTD, sensors send messages to erase the back pointers stored in its subsequent sensors. Also, messages are sent during the search for back pointers in load balancing process. The messages will travel along four directions and the boundaries of the ROI.

Besides, sensors periodically transmit "Hello" messages containing their information (e.g., position, id, color, back pointer stored) to their neighbors in both BTD and EBD. Sensors learn the condition of their neighbors by listening to the "Hello" messages. Because sensors in both algorithms do so, the "Hello" messages will not be counted when comparing the number of sensor messages. Figure 4.6 illustrates how SM varies with *m*.

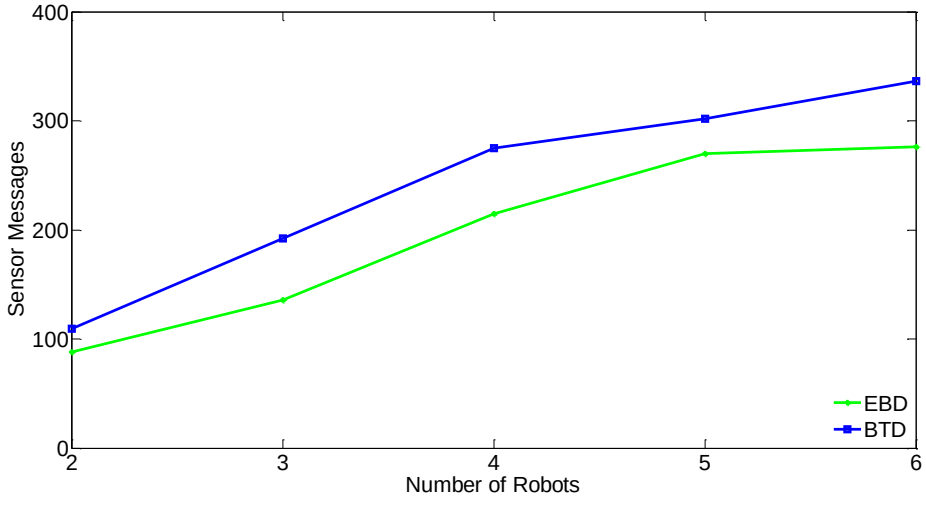


Figure 4.6 Sensor Messages vs Number of Robots

In EBD, SM increases as the number of robots increases, since robots are more likely to get stuck as their number grows. In BTD, the number of sensor messages is higher than that in the case of EBD; moreover, there is also an ascending trend in the number of sensor messages, which is probably due to backtracking and load balancing, since searching for white or gray sensors along the boundaries of the ROI will require large number of sensor messages.

4.2 Coverage Maintenance

Since BTD does not maintain coverage after all the grid points in the ROI get fully occupied with sensors, it cannot be considered for comparison. However, the hole fixing capability of EBD will be tested by comparing it with DDMA, one of the three algorithms proposed by Mei et al. [13], after some adjustments.

4.2.1 Simulation Environment

The environment is the same as that in Section 4.1.1, which is shown in Figure 4.1. The hole fixing algorithms will start after the completion of the sensor deployment phase which may have created sensing holes in the ROI. Robots need to patch the remaining sensing holes from the deployment phase.

For simulation, we adjusted the DDMA slightly as follows. Information stored in sensors belonging to subareas is updated every time a hole is covered. Each sensor stores the distances from itself to all robots and chooses the nearest one as its manager. Robots will send messages to four geographical directions once they finish patching a hole, and the messages will spread through the whole ROI. Sensors update their managers after hearing the messages. Once a sensing hole arises, the gray sensors will compare the distances from their current positions to their managers first, the one holds the shortest distance will send a failure report to its manager which is chosen as the maintainer.

During the simulation, we generated ten sensing holes whose sizes vary from 1 to 6 randomly, with each hole randomly appearing anywhere in the ROI at any time during the simulation period.

The evaluation parameters are different with the ones used in the sensor deployment phase. They are described below.

1. The *Robot Message (RM)* and *Sensor Messages (SM)* remain the same.
2. *Robot Moves (MV)*. It has a different meaning (i.e., average number of moves) in the sensor deployment phase, which accounts for the total number of moves by all robots during hole fixing.
3. *Coverage Time (CT)*. It is the total time for robots to patch all holes.

Here, all the parameters should be minimized since they are proportional to energy, communication and time consumption.

4.2.2 Robot Moves

Robot moves (*MV*) reflects the energy consumption of robots during hole patching directly.

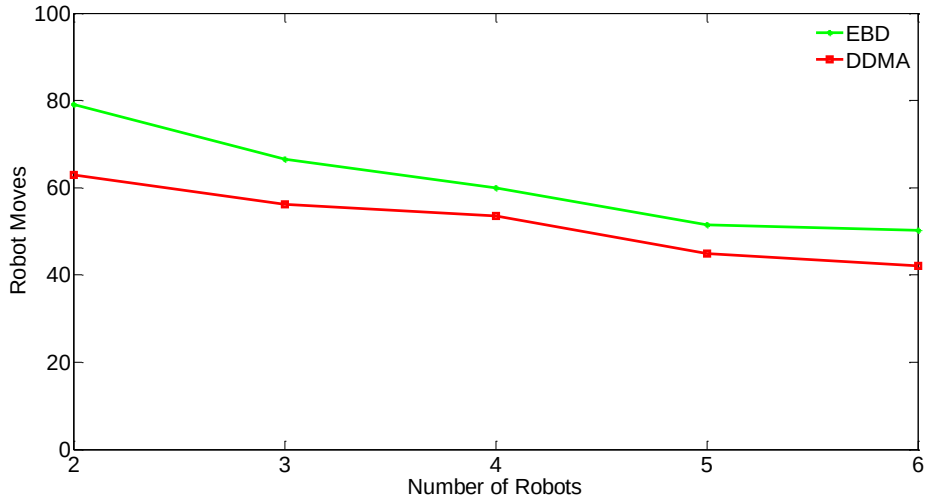


Figure 4.7 Robot Moves vs Number of Robots

Figure 4.7 illustrates how *MV* behaves as *m* increases. For both cases, we notice slight decrease in *MV* as *m* increases. This is due to the fact that the distance from a hole to its worker or maintainer will decrease with the increase in number of robots in the ROI.

The reason that EBD exhibits higher *MV* than its counterpart DDMA is that, the robot move consists of three parts in EBD: (i) the distance of moving to the central points after fully covering the ROI initially, (ii) moving to the holes and patching

them, (iii) returning from the holes to the central points after patching. Thus, the total number of robot moves in case of EBD is higher than that of DDMA.

4.2.3 Robot Messages

Robots in EBD send messages to all the sensors they deploy when they update central points, which happens when they change their subareas. However, in DDMA, robots only send messages to update subareas once they finish fixing a hole.

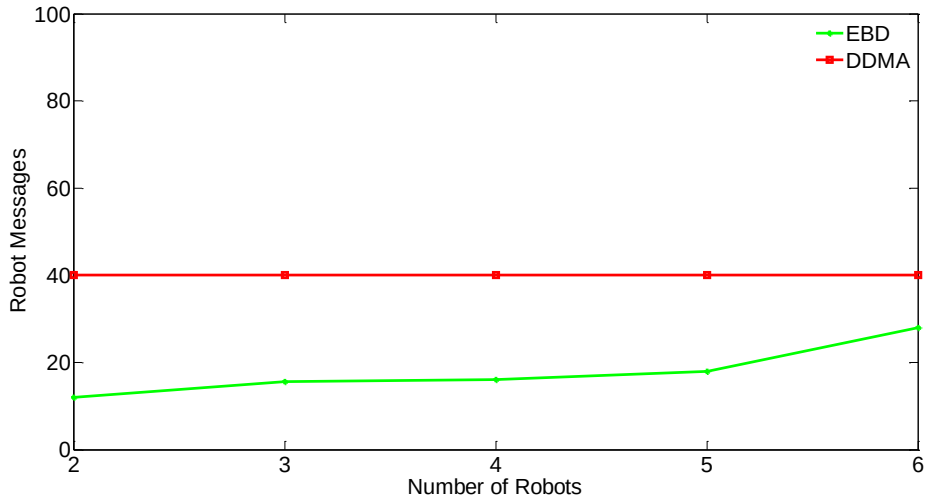


Figure 4.8 Robot Messages vs Number of Robots

Figure 4.8 shows how *RM* varies with *m*. For DDMA, *RM* remains at forty since robots only send message in four geographical directions after finishing patching a hole. The number of holes are assigned as ten, thus the total number of robot messages, which is forty, does not vary with *m*.

While for EBD, we can notice there is an ascending trend in *RM* as *m* increases. More robots there are in the ROI, the smaller each subarea is. As we know, smaller regions are less stable than larger ones, whose central point is more likely to change due to the adjustment of area, resulting in higher number of robot messages.

4.2.4 Sensor Messages

SM in EBD are transmitted among sensors when central points are updated, and also when gray sensors send failure reports to robots; whereas in DDMA, when a robot finishes covering a hole, it will send messages in four directions to inform all sensors about its new position. The messages will spread within the ROI.

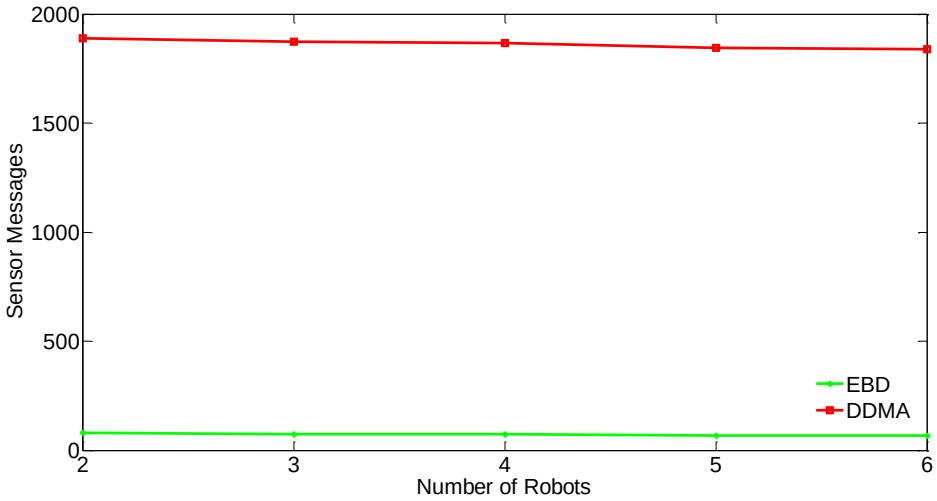


Figure 4.9 Sensor Messages vs Number of Robots

Figure 4.9 illustrates the relationship between *SM* and *m*. In this case, we notice a descending trend for DDMA and a nearly flat trend for EBD. For DDMA, the number of *SM* is high since the robots send messages to inform all the sensors in the ROI with their new position once they cover a hole. Compared with DDMA, EBD has much lower sensor message consumption.

4.2.5 Coverage Time

CT reflects the summation of time for patching all holes in the ROI.

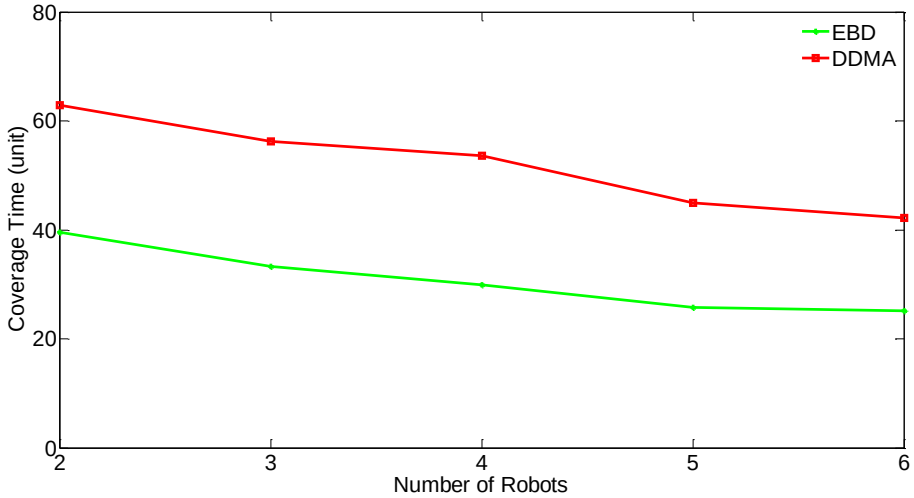


Figure 4.10 Coverage Time vs Number of Robots

Figure 4.10 demonstrates the relationship between *CT* and *m*. We can observe decreasing trends in both approaches since the increase of *m* shortens the distance between robots and holes. In this case, EBD outperforms DDMA by 20 time units approximately.

Chapter 5: Conclusions and Future Work

5.1 Conclusions

This thesis provided a solution for sensor deployment and coverage maintenance problem in multi-robot scenario in failure-prone environment, which we call the Election-Based Deployment (EBD) approach. A localized sensor placement algorithm was proposed firstly by extending the existing Back-Tracking Deployment (BTD) approach. We considered different strategies for robots according to their starting positions to provide regular subareas for the coverage maintenance algorithm. We also adjusted robot reaction to the dead end situation; they will move to the nearest white sensor instead of the back pointer. We can observe from the simulation results that the proposed EBD balances the workload of robots and reduces the communication overhead to a certain extent.

A hole-fixing algorithm was proposed following the sensor placement algorithm as an extension in which the ROI is divided into several subareas, each having at least one robot. The process of selecting the most suitable robot to cover a hole resembles like an election.

EBD is the first carrier-based localized algorithm that is able to achieve full coverage with multiple robots in failure-prone environment since it combines both sensor deployment and coverage maintenance process.

5.2 Future Work

Some future works can be done in order for the improvement and extension of our work. For some challenging scenarios such as the high probability of sensor failure and communication failure in hazardous environment, the proposed protocol still needs to be improved for better performance. In future, we intend to extend our proposed algorithm in the following ways.

EBD is designed under the assumption of unlimited sensor load, which is not practical in the real environment. The scenario of robot reloading should be taken

into consideration. A feasible solution for this issue is mentioned in [22], which introduces a group of robots in charge of transporting sensors to the builders.

Another possible improvement can be to reduce the energy and communication consumption, which can start by looking for another white sensor searching method, since most of the messages are for searching white sensors.

The model of EBD is established on a two-dimensional map which is unrealistic in real world, since it is inevitable for a large scaled ROI to have some altitude difference. Adjustment of the algorithm to fix three-dimensional environment should be investigated.

References

[1] M. Ma and Y. Yang, "Adaptive triangular deployment algorithm for unattended mobile sensor networks," *IEEE Trans. Comput.*, vol. 56, no. 7, pp. 946-958, 2007.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102-114, 2002.

[3] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, W. J. Kaiser, and H. O. Marcy, "Wireless integrated network sensors: low power systems on a chip wireless integrated network sensor," in *Proc. 24th European Solid-State Circuits Conf.*, 1998, pp. 9-16.

[4] N. Bartolini, T. Calamoneri, E. G. Fusco, A. Massini, and S. Silvestri, "Snap and spread: a self-deployment algorithm for mobile sensor networks," *Lecture Notes in Comput. Sci.*, vol. 5067, pp. 451-456, 2008.

[5] X. Li, A. Nayak, D. Simplot-Ryl and I. Stojmenovic, "Sensor placement in sensor and actuator networks," in *Proc. Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Commun., Wiley*, 2010, ch. 10, pp. 263-294.

[6] Wireless integrated networks sensors project [Online]. Available: http://www. janet.ucla. edu/WINS/

[7] R. Min, M. Bhardwaj, N. Ickes, A. Wang, and A. Chandrakasan, "The hardware and the network: total-system strategies for power-aware wireless microsensors," in *Proc. IEEE CAS Workshop Wireless Commun. and Networking*, Sep, 2002.

[8] The ultra low power wireless sensor project [Online]. Available: http://www. mtl.mit.edu/jimg/ project_top.html

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

[10] C. Chang, Y. Chen, and H. Chang, “Obstacle-resistant deployment algorithms for wireless sensor networks,” *IEEE Trans. Veh. Technol.*, vol. 58, no. 6, pp. 2925-2941, 2009.

[11] X. Li, G. Fletcher, A. Nayak, and I. Stojmenovic. “Placing sensors for area coverage in a complex environment by a team of robots”, *ACM Trans. Sensor Networks*, vol. 11, no. 1, pp. 3-11, 2014.

[12] I. Mezei, V. Malbasa and I. Stojmenovic, “Robot to robot communication aspects of coordination in robot wireless networks,” *IEEE Robot. Automat. Mag.*, pp. 63-69, Dec, 2010.

[13] Y. Mei, C. Xian, S. Das, Y. C. Hu, and Y. H. Lu, “Sensor replacement using mobile robots,” *Comput. Commun.*, vol. 30, no. 13, pp. 2615-2626, 2007.

[14] G. Fletcher, “Sensor placement and relocation by a robot team,” M.S. thesis, Dept. Elect Eng. Comput. Sci., Univ. of Ottawa, Ottawa, ON, 2010.

[15] E. Biagioni and K. Bridges, “The application of remote sensor technology to assist the recovery of rare and endangered species,” *Intel J. High-Performance Comput. App.* (Special issue on distributed sensor networks), vol. 16, no. 3, pp. 112-121, 2002.

[16] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, “Habitat monitoring: application driver for wireless communications technology,” in *Proc. ACM SIGCOMM Workshop Data Comm. in Latin America and the Caribbean*, 2001.

[17] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, “Wireless sensor networks for habitat monitoring,” in *Proc. 1st ACM Intel Workshop WSN and App.*, 2002.

[18] S. Chessa and P. Santi, “Crash faults identification in wireless sensor networks,” *Comput. Comm.*, vol. 25, no. 14, pp. 1273-1282, 2002.

[19] X. Li, N. Santoro, and I. Stojmenovic, “Localized distance-sensitive service discovery in wireless sensor and actor networks,” *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1275-1288, 2009.

[20] I. Mezei, M. Lukic, V. Malbasa, and I. Stojmenovic, “Auctions and iMesh

based task assignment in wireless sensor and actuator networks," *Comput. Commun.*, vol. 36, no. 9, pp. 979-987, 2013.

[21] X. Fan, Z. Zhang, and H. Wang, "The probabilistic sense model for coverage hole elimination in WSN," in *Proc. 33rd Chinese Control Conf.*, 2014, pp. 422-427.

[22] M. Batalin and G. Sukhatme, "Coverage, exploration and deployment by a mobile robot and communication network," *Telecommun. Syst.*, vol. 26 no. 2-4, pp. 181-196, 2004.

[23] C. Chang and J. Sheu, "An obstacle-free and power-efficient deployment algorithm for wireless sensor networks," *IEEE Trans. Syst. Man. Cybern.*, vol. 39, no. 4, pp. 795-806, 2009.

[24] G. Fletcher, X. Li, A. Nayak, and I. Stojmenovic, "Back-tracking based sensor deployment by a robot team," in *Proc. IEEE Conf. Sensor, Mesh Ad Hoc Communication and Networks*, 2010, pp. 1-9.

[25] P. E. Hart, N. J. Nilsson, and B. Raphael, "Formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100-107, 1968.

[26] 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.

[27] X. Li, H. Frey, N. Santoro, and I. Stojmenovic, "Strictly localized sensor self-deployment for optimal focused coverage," *IEEE Trans. Mob. Comput.*, vol. 10, no. 11, pp. 1520-1533, 2011.

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

[29] R. Falcon, X. Li, and A. Nayak, "Carrier-based coverage augmentation in wireless sensor and robot networks," in *Proc. IEEE Distributed Computing System Conf.*, 2010, pp. 234-239.

[30] X. Li, H. Frey, N. Santoro, I. Stojmenovic, "Focused-coverage by mobile

sensor networks," in *Proc. IEEE 6th Int. Conf. on Mobile Ad hoc and Sensor Systems*, 2009, pp. 466–475.

[31] 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.

[32] 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.

[33] H. Li, A. Barnawi, I. Stojmenovic, and C. Wang, "Market-based sensor relocation by robot team in wireless sensor network," *Ad Hoc & Sens. Wireless Networks*, vol. 22, pp. 259-280, 2014.

[34] M. P. Johnson, D. Sariöz, A. Bar-Noy, T. Brown, D. Verma, and C. W. Wu, "More is more," *ACM Trans. Sens. Networks*, vol. 8, no. 3, pp. 1-19, 2012.

[35] 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.

[36] 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.

[37] J. Oyekan and H. Hu, "Biologically-inspired behaviour based robotics for making invisible pollution visible: a survey," *Advanced Robot.*, vol. 28, no. 5, pp. 271–288, 2014.