uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Gregory Paul Fletcher
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Placement and Relocation of Wireless Sensor Nodes by a Team of Robots
TITRE DE LA THÈSE / TITLE OF THESIS

A. Nayak
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

I. Stojmenovic
CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

J. Zhao

Chung-Horng Lung

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Placement and Relocation of Wireless Sensor Nodes by a Team of Robots

Greg Fletcher

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Master of Computer Science degree

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering (SITE)
Faculty of Engineering
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5

© Gregory P. Fletcher, Ottawa, Canada, 2010

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-74143-6
Our file Notre référence
ISBN: 978-0-494-74143-6

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Acknowledgements

I would like to acknowledge my sincerest gratitude to my supervisor Prof. Dr. Amiya Nayak of Ottawa-Carleton Institute for Computer Science. Without his enthusiastic and expert guidance during writing this thesis, it would not have been possible to complete this thesis. I am grateful to my co-supervisor Prof. Dr. Ivan Stojmenovic for the help in preparation of this thesis. I would also like to thank Dr. Xu Li for the valuable contributions and discussions in writing this thesis and associated papers. I would also like to thank my friends and my family in particular for their continuous support.

Table of Contents

List of Figures

Abstract

We propose a novel localized sensor placement algorithm for wireless sensor networks, named *Back-Tracking Deployment* (BTD). Through simulation we show that BTD far outperforms the only competing algorithm LRV [2] in robot moves and robot messages at no additional cost in a failure-free environment and at minimal cost of sensor messages in a failure-prone environment.

Further, we propose the first localized carrier-based sensor relocation algorithm for wireless sensor networks, named *Randomized Robot-assisted Relocation of Static Sensors* (R3S2). We also presented a grid-based variant, called G-R3S2 and a cluster-based variant, called C-R3S2 and C-G-R3S2.

Through extensive simulation, the effectiveness and practicality is shown. The simulation results indicate in particular that robots following the LRV policy (G-R3S2 and C-G-R3S2) outperformed those that did not across all measured metrics. Further, the cluster-based variants outperformed their counterpart (C-G-R3S2 vs. G-R3S2 and C-R3S2 vs. R3S2) across all metrics.

Chapter 1: Introduction

1.1 Background Information

A Wireless Sensor Network (WSN) is typically made up of a large number of small, inexpensive and resource-constrained sensing devices, which are powered by low-energy batteries and connected via wireless communication links. The primary goal for a WSN is low-level surveillance and data gathering in a given region of interest (ROI) for various applications such as traffic analysis, habitat study and intrusion detection. For this reason sensors must fully cover the ROI without internal sensing holes. At times further requirements such as node degree [37], node density [17] or coverage focus [25] may apply. However, it cannot be expected that sensors are placed in a desired way as they are often dropped in a stochastic manner due to factors such as inaccessibility to the ROI or tight deployment budget.

Recent research has examined controlled mobility and how it can be used to improve coverage in a WSN. In this case sensors are equipped with some form of mobility and after the initial random deployment, can self-deploy to achieve full coverage over the ROI. In an attempt to reduce deployment costs, Wireless Sensor and Robot (also called Actuator or Actor) Networks (WSRN) [36] have also been examined. Rather than equipping each individual sensor with mobility, a WSRN employs small number of mobile robots to assist in the deployment process. The robots and sensors communicate via wireless links. In this network model, robots are considered to be resource-rich as compared to sensors. The robots are typically involved in making decisions and performing actions on themselves (i.e. controlled movement), as well as on the sensors (i.e. repairing failed sensors).

In the *self-deployment* approach, sensors autonomously and intelligently change their geographic location, resulting in a more desirable distribution. Alternatively, mobile robots may carry static sensors as payload and place the sensors at pre-determined locations (i.e. vertices on a geographic graph). This approach is known as *carrier-based* sensor deployment. In this deployment method the robots move throughout the ROI and place the sensors in an intelligent way, such that the entire ROI will be covered by the sensors' sensing radii upon completion of the deployment. In this thesis we will focus on

localized solutions to carrier-based sensor deployment, as well as carrier-based sensor relocation to improve coverage in the event of sensor failures within the ROI. We will see the previous research into carrier-based deployment is limited, while localized solutions to carrier-based sensor relocation have yet to be studied.

1.2 Problem Statement

1.2.1 Sensor Placement

As previously stated, sensors are typically dropped – from an airplane for example – in a stochastic manner. This leads to imperfect coverage. Assuming that the sensors are equipped with some form of mobility, the sensors can self-relocate to improve the area coverage. We will consider, however, the case where the ROI is accessible, thus allowing for a more intelligent placement strategy. In this case mobile carriers move throughout the ROI and place static sensors at appropriate intervals; thus removing the strong assumption that all sensors must be equipped with mobility. The following is a formal description of the carrier-based sensor placement problem.

A number of mobile robots, pre-loaded with static sensors, are scattered randomly in an unknown bounded two-dimensional ROI (i.e. terrain and boundary information are not known by robots a priori). Robots are aware of their position by attached GPS or other positioning techniques [20] and are able to detect obstacles and ROI boundaries by attached scanning laser range-finder, for example. Sensors are assumed to be homogeneous, with sensing radii, r_s, and communication radii, r_c (also the same for robots). Sensors and robots are able to communicate directly if they are within the given communication range. Supposing that each robot is carrying a sufficient number of sensors, the goal is to develop an algorithm for robots to move and drop sensors to construct a connected sensor network fully covering the ROI.

The algorithm should enable robots to avoid physical obstacles during sensor placement. Because physical movement consumes a large amount of energy, the algorithm is expected to yield a minimal number of robot moves. To save bandwidth and energy, the algorithm should reduce communication to a minimum as well. Thus, we seek localized solutions, which rely only on some available local knowledge, without resorting to global network information.

1.2.2 Sensor Relocation

A number of mobile robots are scattered randomly throughout a bounded ROI – whose boundaries are known a priori – with sensors also already randomly deployed. As with the sensor placement problem, robots are aware of their position by attached GPS and are able to detect physical obstacles and ROI boundaries. Again, sensors and robots may communicate directly assuming that they are within each other's communication radius.

A continuous uncovered area in the ROI is referred to as a *sensing hole* (or hole for simplicity). Although the ROI may not be fully covered, some sensors could be redundant from a local perspective. To conserve energy, these sensors remain in a "sleeping" mode, and are thus called *passive sensors*. Non-redundant sensors are essential for coverage. They remain alert and are thus referred to as *active sensors*.

Consider a special case that there is a redundant sensor at any point along a robot's trajectory and that every sensing hole is small enough to be patched by a single sensor. In this artificial scenario, a robot's task is to find the shortest tour that visits every sensing hole exactly once. This is the well-known NP-complete Travelling Salesman Problem [35]. Hence, the carrier-based sensor relocation coverage repair problem is NP-hard, and no optimal solution can be found in polynomial time even with global knowledge and centralized control.

The goal is to develop an effective and efficient sensor relocation algorithm, by which mobile robots find and collect passive sensors and deliver them to reported holes. Here, effective means high repair rate and efficient refers to minimal repair delay with minimal communication overhead.

1.3 Existing Solutions

The key problem associated with carrier-based sensor placement is that the carriers must fully explore the ROI in order to construct full coverage. This is related to the map exploration problem in the traditional mobile robot field. Map exploration has long been studied for both single robot scenarios [42] and for multiple robot scenarios [4]. However, these algorithms are not applicable to WSRN due to model differences.

For example, in a mobile robot system, robots typically have vision but no communication and make protocol decisions by observing other robots' movements. These algorithms also typically require unrealistic perception range – each robot is able to see the entire environment, which is potentially very large, for example. Alternatively, in a WSRN, robots typically have some communication capabilities, but no vision.

Movement-assisted sensor deployment was studied in the literature in a very limited context. The majority of the previous works address sensor self-deployment for mobile sensor networks. Few shed light on the carrier-based deployment problem. In this section, we will review the carrier-based algorithms in brief. Each of the reviewed algorithms have major deficiencies such as inefficient communication, unclear or lack of terminating conditions and/or lack of full coverage guarantee, even in a failure-free environment. We will review these algorithms in more detail in Chapter 2.

Chang et al. presented a carrier-based sensor deployment algorithm in which a robot would move in a snake-like manner (SLD), deploying sensors at pre-determined intervals such that full coverage would be achieved [6, 7]. However, the authors did not present clear conditions under which the algorithm terminates. Further, it can be shown with a simple example that the algorithm fails to provide full coverage, even in an ideal, failure-free environment (see Sec. 2.1 for example).

Batalin and Sukhatme presented a carrier-based deployment algorithm following a Least Recently Visited (LRV) movement policy [2]. However, as with SLD, the authors did not provide clear terminating conditions. Additionally, the authors only presented the algorithm for a single robot scenario, which is not practical for larger network sizes. Finally, we will show in Chapter 5 that while LRV provides full coverage, it requires a large number of wasted movements and messages.

The sensor relocation algorithm proposed in this thesis is the first to offer a localized solution for robot-assisted sensor relocation for coverage repair in a static WSN. As with sensor deployment, existing research studying movement-assisted sensor relocation aims to solve a different problem, sensor self-relocation, which requires that all sensors have locomotion. This is a very strong requirement and can lead to high deployment costs.

1.4 Motivations and Objectives

As an alternative to stochastic node dropping, we expect that a carrier-assisted sensor placement algorithm should be localized – in order to keep message cost to a minimum. Additionally, the algorithm should support multiple robots dropping sensors, as well as the ability to handle single or multiple node failures. A solution for the carrier-based sensor relocation problem should also be a localized algorithm. It should also support multiple robots acting as network maintainers.

As previously stated most of the existing solutions require that every sensor be equipped with locomotion and/or adopt a centralized approach. The SLD and LRV algorithms are the only proposed localized solution to the carrier-based sensor deployment problem. However, as previously explained both of these algorithms are incomplete. No localized solutions exist to solve the problem of carrier-based sensor relocation.

We describe the design of new localized algorithms which solve the carrier-based sensor deployment (BTD) problem, as well as the novel carrier-based sensor relocation (R3S2) problem. The proposed algorithms are expected to outperform existing solutions in terms of message cost and deployment/repair latency, while providing clear terminating conditions and coverage guarantee in the case of sensor deployment.

1.5 Assumptions

The WSN referred to in this thesis will be composed of one or more mobile robots and a set of homogeneous sensors. The sensors are assumed to be static. The communication radius is assumed to be the same for all sensors, as is the sensing radius. The communication radius is assumed to be at least twice the sensing radius. The communication and sensing radii are modeled with the unit disk graph (UDG) and any two nodes may communicate directly if their separation is less than the communication radius. As a result, all links are assumed to be bi-directional.

Ideal physical and MAC layer are used, as in the existing solutions in literature [2, 6, 7]. Packet transmissions have no collisions and no failures. Message transmission

within the WSN is assumed to be asynchronous. Therefore, all messages will eventually reach the desired destination; however, there are no guarantees on message delay.

For the sensor placement algorithm, robots are assumed to be able to carry enough sensors to fully cover the ROI. For the sensor relocation algorithm, there is no assumed limit on the number of sensors that robots are able to pick up and carry. Robots are also assumed to have sufficient energy source to continually move throughout the ROI and communicate with previously deployed sensors until the sensor placement or relocation is complete.

1.6 Contributions

1.6.1 Sensor Placement

We present a novel localized solution, named *Back-Tracking Deployment* (BTD), to the above stated problem. BTD introduces the concept of back-tracking in order to recover from encountered dead ends. Further, it supports multiple robots moving throughout the ROI, placing sensors simultaneously.

Through extensive simulation, the performance of BTD in comparison with SLD [6, 7] and LRV [2] is evaluated. SLD does not support multiple robots or tolerate sensor failures. Thus, the comparison is strictly for failure-free single-robot scenarios. In our examples, SLD generates 40% - 50% coverage over the ROI on average, with a variation from 20% to 80%. Alternatively, BTD guarantees full coverage in failure-free environments. LRV was designed for single robot scenarios, and it does not terminate by itself. In our simulation, we extended LRV to support multiple robots and we consider it terminated once the ROI is fully explored. The simulation results show that, compared with LRV, BTD has > 80% savings in robot moves and robot messages at no cost in a failure-free environment and at minimal cost of < 0.4 messages per sensor and < 8% sensing coverage in the presence of failures.

1.6.2 Sensor Relocation

We present a localized algorithm, named Randomized Robot-assisted Relocation of Static Sensors (R3S2), along with grid and cluster-based variants G-R3S2 and C-R3S2

(C-G-R3S2), respectively. R3S2 is the first localized algorithm which addresses the problem of carrier-based sensor relocation.

Through extensive simulation, the performance of the above algorithm and its variants based on repair delay, repair rate and message cost by varying the number of sensors, holes and robots is evaluated. Across all metrics the algorithms which employed the grid-based movement policy outperformed their non-grid counterpart (i.e. G-R3S2 vs. R3S2 and C-G-R3S2 vs. C-R3S2). Additionally, the introduction of the cluster-based approach provided a noticeable improvement over the variants that did not use clusters.

1.7 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 provides a literature review on sensor self-deployment algorithms, robot-assisted sensor deployment algorithms and sensor self-relocation algorithms. Chapter 3 proposes the Back-Tracking Deployment (BTD) algorithm. Chapter 4 proposes the Randomized Robot-assisted Relocation of Static Sensors (R3S2) algorithm. Chapter 5 presents simulation results, followed by conclusions in Chapter 6.

Chapter 2: Literature Review

2.1 Carrier-Based Sensor Deployment

Chang et al. presented a deterministic, snake-like deployment approach (SLD) [6, 7]. Following the SLD approach, a robot moves step by step along a pre-computed graph starting from the North-West corner of the environment, leading to optimal coverage and drops a sensor after each step. With differently defined vertex selection rules, the robot trajectory will exhibit different patterns, such as an 'S' shape [6] or a spiral shape [7].

SLD is prone to getting stuck at dead ends due to obstacles or previously deployed sensors, as discussed in [28]. Since dead end recovery was not discussed, we can conclude that the algorithm does not guarantee full coverage. Additionally, the algorithm does not support multiple robots nor do the authors discuss a strategy for repairing emerging sensing holes. Shiu [38] induced the lack of coverage guarantee for this algorithm to concave regions and suggested to treat each concave region as a separate environment. The obstruction caused by obstacles and previously deployed sensors is overlooked and the coverage guarantee is still not accomplished.

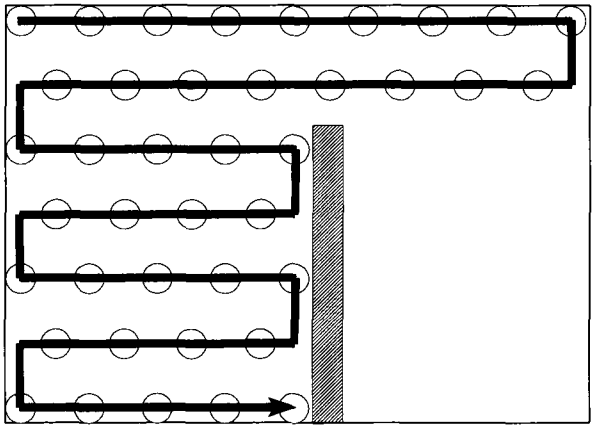


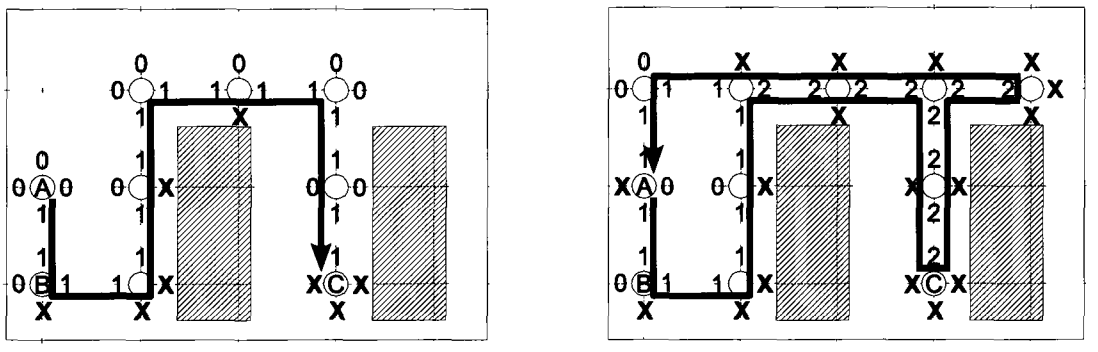
Figure 1. Shortcoming of SLD Algorithm

This lack of dead end recovery leading to incomplete coverage following the SLD algorithm can be seen in Fig. 1. In this simple scenario with only a single obstacle, the robot is unable to construct full coverage, with the South-East region of the environment

unexplored. Further, SLD requires some prior knowledge of the ROI – namely, the ROI boundary information – in order for the robot to start in the North-West corner of the ROI. However, this information may not be known a priori. Additional deficiencies of SLD were discussed in Sec. 1.3.

Batalin and Sukhatme proposed the Least Recently Visited (LRV) algorithm [2]. Sensors deployed within the ROI store *weights* in each direction that a robot can move. The weights represent the number of times that a robot has traversed the given direction – either coming or going – for a particular sensor. The sensors recommend the direction with the lowest weight (i.e., the least recently visited direction) for the robot's next move.

At initiation, the robot places a sensor at its current location. When a new sensor is placed, the weight in every direction is set to zero initially. After each movement by the robot, sensors update their weights accordingly. In order to determine its next movement, robots listen for messages from the sensor at their current location. Each direction is also given a rank, therefore in the event of a tie only one direction is recommended. If a direction is obstructed by a physical obstacle or ROI boundary, the robot will request the next least recently visited direction from the sensor at its current location.



(a) Dead End (b) Full Trajectory

Figure 2. Least Recently Visited Approach

Fig. 2 illustrates the LRV approach. In this case the robot starts from position *A* and is able to travel in the four geographic directions: North, East, South and West. The numbers surrounding deployed sensors indicate the weight for a given direction, while the Xs indicate that a particular direction is known to be obstructed – either by an

obstacle or ROI boundary. The thick arrowed lines indicate the robot's trajectory as it explores the environment. Fig. 2(a) shows the robot's trajectory from A to the dead end at C. Fig. 2(b) illustrates the robot's recovery from the dead end, while eventually providing full sensing coverage before returning to position A. The algorithm was only presented with a single robot. A multi-robot extension was discussed, but for a different purpose. Further shortcomings were discussed in Sec. 1.3.

2.2 Sensor Self-Deployment

2.2.1 Virtual Force Approach

Applying virtual forces to mobile sensors has been studied thoroughly for the purpose of sensor self-deployment. The idea is simple in principle. Each node in the network, i, computes a virtual force contribution from each of its neighbours, F_{ij}. The node then moves according to the vector summation, F_i, representing the total force acting on the given node. The virtual force approach has been applied to various physical models such as potential fields, molecules and electro-magnetic particles. The goal is to achieve uniform sensor distribution, providing full coverage within the ROI.

Howard et al. proposed a potential field-based approach to construct a connected WSN which provides full coverage over the ROI [22]. In this approach sensors receive force contributions from both neighbouring sensors and obstacles forcing sensors to move apart, fully covering the area in the process. Each sensor locally computes the total force acting upon it using physical laws. Once the total force is computed the sensors move according to the calculated force vector.

In order to guarantee termination and prevent *node oscillation* – a node moving slightly then returning to its previous position – the authors include a friction force which opposes movement. Over time the friction force causes node velocity to reach 0, bringing the system to *static equilibrium*. By guaranteeing static equilibrium, the authors also ensure that node oscillation will eventually be stopped.

Poduri et al. presented a virtual force-based deployment algorithm which seeks to control node position in order to guarantee sensing coverage and k-edge-connectivity within the network [37]. The authors proposed Neighbor-Every-Theta (NET) graphs. NET graphs are defined such that a given node has a neighbour within every θ sector of

its communication range. This condition is illustrated in Fig. 3 [37]. In Fig. 3(a), the node does not satisfy the NET condition as there is a gap greater than θ with no neighbours. Fig. 3(b), however, satisfies the condition and is thus a valid NET graph. NET graphs are shown to have an edge connectivity of at least $\left\lfloor \frac{2\pi}{\theta} \right\rfloor$ when $\theta < \pi$, as a result the graph can produce the desired connectivity based on a single parameter, θ. Recall from Menger's theorem [34], that a graph is said to be k-edge-connected if there exists at least k-edge independent paths between any two nodes in the graph.

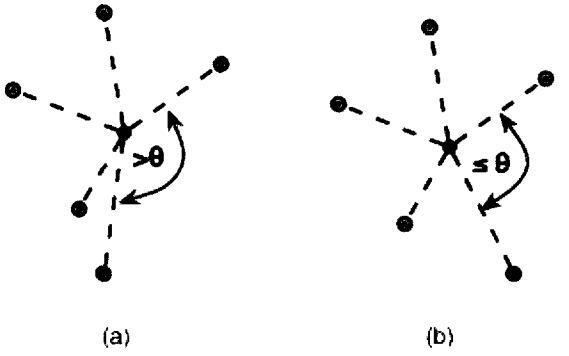


Figure 3. NET Condition

For the deployment algorithm, the authors present an algorithm based on virtual potential fields. Mobile sensing nodes with communication and sensing radii, r_c and r_s, respectively are initially packed tightly. In this way the NET graph and k-edge-connectivity conditions are met trivially. Sensors exert repulsive – F_{repel} – and attractive forces – $F_{attract}$ – on neighbouring nodes. F_{repel} tends to infinity as the distance between two nodes approaches zero. $F_{attract}$ tends to infinity as the distance between two nodes approaches r_c. Because of the initial proximity of the sensing nodes, each node begins to repel its neighbours, increasing the sensing coverage. During this process the number of neighbours for each node begins to decrease as the distance between nodes surpasses r_c. Once the number of neighbours nears the NET condition, a node assigns a priority to its neighbours. Neighbours contributing to a larger sector angle receive higher priority and thus feel attractive forces in order to keep them within the node's communication radius.

Garetto et al. proposed an event-driven self-deployment strategy [17]. The goal of the presented algorithm is to achieve a regular triangle tessellation layout. Upon event occurrence nodes relocate to sample and control the event, while maintaining connectivity. That is, nodes which are further away move closer to the event and nodes which are close to the event may move further out. Once the event ends, nodes return to the regular sensing configuration.

In order to control movement such that the desired configuration is achieved and the node density condition is met when an event occurs, virtual forces are applied. The authors propose the following forces. Exchange forces from neighbouring sensors, potential forces from detected events within the environment and friction force. Therefore during deployment (i.e. when no event has taken place), only exchange forces and friction are considered. A node k exerts an exchange force on another node i if and only if k is a neighbour of i and there does not exist another node k' such that $|k'i| < |ki| \wedge \angle kik' < \frac{\pi}{6}$. This condition ensures that only a maximum of six neighbours can exert forces on node i at a particular instant in time t, and as a result the final network will have a triangle tessellation layout. Exchange forces are repulsive if the nodal separation is less than r_s and attractive otherwise. As with the repulsive and attractive forces in the algorithm from Poduri et al. [37], the attractive exchange force tends to infinity as the nodal separation approaches r_c and the repulsive force tends to infinity as the nodal separation approaches zero. Potential forces can also be attractive or repulsive depending on the detected event intensity. Nodes which are further away will move closer to the event location, while closer nodes will move away from the event epicentre, ensuring that no nodes will be within a certain threshold distance of the epicentre. Finally, the friction force will always oppose movement and is proportional to the node's velocity. Similar to the algorithm presented by Howard et al. [22] the friction force will eventually cause nodes to stop their movement; ensuring that the system will reach static equilibrium and preventing node oscillation.

2.2.2 Point Coverage Approach

Li et al. presented a novel problem associated with sensor self-deployment, achieving *focused coverage* around a point of interest (POI) [24, 25]. The authors also

introduce a new evaluation metric, *coverage radius*. Coverage radius is defined as the radius of the maximal hole-free sensing disc enclosing the given POI. The authors presented two strictly localized algorithms Greedy Advance (GA) and Greedy-Rotation-Greedy (GRG) which solve the focused coverage problem.

The problem was defined as follows. Given a set of mobile sensors with communication range $r_c \geq \sqrt{3}r_s$, construct a connected network around the POI, P, with a triangle tessellation (TT) layout. Sensors are aware of their own position, as well as the position of P. The reason for the TT layout is that it maximizes the coverage area of a given number of nodes without sensing holes when sensing nodes are placed on the vertices [1, 32, 43]. Additionally it guarantees network connectivity when $r_c \geq \sqrt{3}r_s$.

GA works by having nodes continually move greedily toward the location of P. Each node attempts to move to a TT vertex that is closer in distance to P. The authors show that for a TT layout there will be a maximum of only two possible such vertices. For a sensor that is in a corner position there will only be one possible vertex to move to. To avoid collisions the authors define a set of three rules controlling movement. The first rule assigns priority to prevent two neighbouring nodes from simultaneously advancing to the same location. The second rule forbids certain movements in order to avoid two non-neighbours from simultaneously advancing to the same vertex. The rule states that a node at vertex $\langle i + 1, j, 1\rangle$ cannot move to vertex $\langle i, j, 0\rangle$. The third rule states that any node that is adjacent to P can move to P, as long as P is not occupied.

Fig. 4 illustrates the hop selection in GA [25], based on six nodes and their possible next hops. In this example, the possible next hops are highlighted by thick arrows. Either node 1, 2 or 3 will move to P based on the priority rule, described above. Again due to the priority rule, the next hop of nodes 4 and 5 or nodes 5 and 6 will ensure that no collision occurs. Because nodes 4 and 6 are not neighbours it is possible that a collision may occur if both nodes decide to move to the vertex $\langle 4, 1, 0\rangle$. However, due to the forbiddance rule, node 4 cannot move to the vertex $\langle 4, 1, 0\rangle$ (illustrated with a lighter arrow). As a result, no collision will occur.

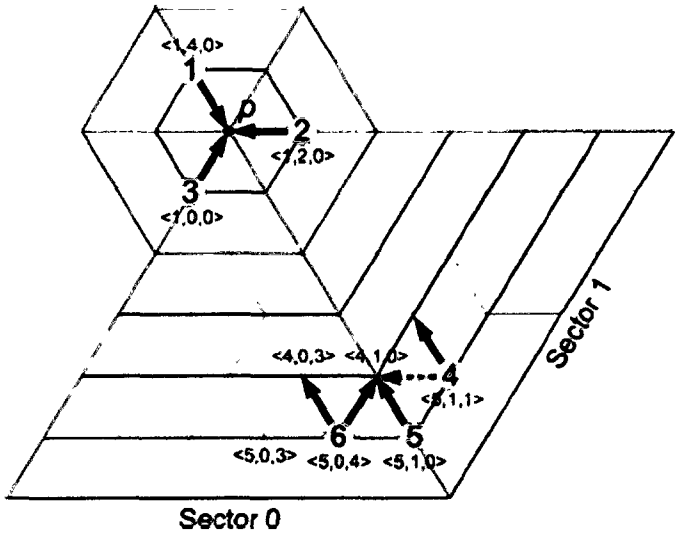


Figure 4. Hop Selection – Greedy Advance

GRG extends GA by adding a new movement type, *rotation*. Rotation is used when a sensor's greedy advance movement is blocked. The rotating sensor moves in a pre-determined direction – either clockwise or counter-clockwise – along the same layer. Rotation stops once the sensor reaches a vertex where greedy movement can resume or it reaches the vertex where rotation began. Due to the asynchronous nature of the algorithm, it is possible that a rotating node may block a greedy movement. To overcome this situation the authors introduce a suspension rule in which a rotating sensor in layer $i - 1$ will suspend its rotation if it detects a neighbour in layer i is also rotating. In this way the sensor in layer i will rotate past the suspended sensor and will then be able to perform a greedy movement. If a sensor performing greedy advance and a rotating sensor are competing for the same vertex, priority will be given to the greedy advance movement. Further rules were introduced for special cases. Fig. 5 illustrates the hop selection in GRG [25], based on six nodes and their possible next hops. In this example greedy advance next hops are shown as black arrows, while rotations are shown as blue arrows. The authors prove that both GA and GRG terminate in finite time, yielding a connected network, with maximized hole-free coverage.

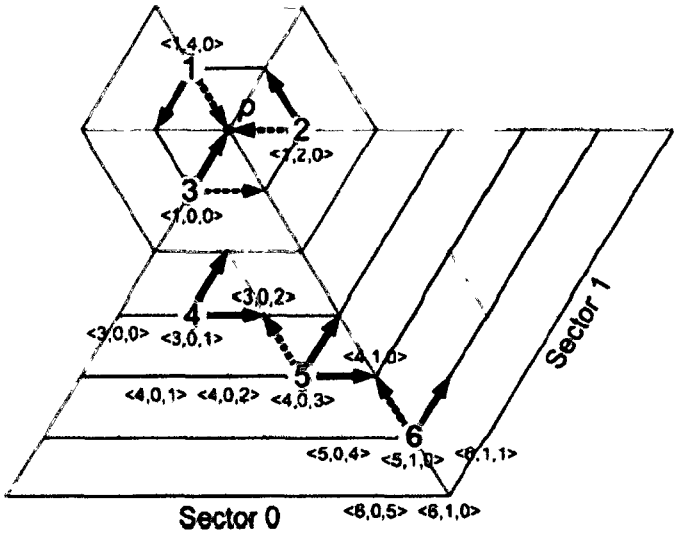


Figure 5. Hop Selection – Greedy Rotation Greedy

Li et al. presented an optimized version of GRG, called OGRG in [26]. OGRG uses deployment polygons as opposed to hexagons for sensor rotation. The reason for this decision is that deployment polygons provide the best approximation to circles. OGRG provides guaranteed circular coverage radius maximization. GRG and GA had been presented assuming an obstacle-free environment. However, Li et al. removed this assumption and presented an obstacle avoidance strategy, called GRG with Obstacle Penetration (GRG/OP) [27]. The obstacle avoidance algorithm enables mobile sensors to behave like water molecules in a U-tube when obstructed by physical obstacles. Further, the authors show that by treating the ROI boundaries as obstacles and the geographic centre of the ROI as the POI, GRG/OP can be used to solve the traditional area coverage problem in addition to the focused coverage problem that GRG was initially designed to solve.

2.2.3 Incremental Approach

Howard et al. presented an incremental approach for sensor self-deployment algorithm [21]. In this algorithm sensors are assumed to have vision and are deployed one at a time with an added *visibility constraint*. That is, every sensor must be *visible* to at least one other sensor. The goal is to construct a connected network while maximizing the total visible area. The algorithm works in the following way.

At initiation only one node is assumed to be deployed. This node acts as the network starting point. The algorithm runs in a series of iteration deploying one sensor per iteration until all available sensors have been deployed. The deployment location for the sensors is chosen by a central controller based on the information provided by sensors already deployed. The authors separate each iteration into three phases, *selection*, *assignment* and *execution*.

In the selection phase the central controller selects the location for the next sensor to be deployed. In the first step of the selection phase, the controller builds an occupancy grid, in which every cell is *free*, *occupied* or *unknown*. A cell is free if it is known to contain no obstacle. A cell is occupied if it is known to contain an obstacle and a cell is unknown otherwise. Once the occupancy grid is computed the controller than computes a configuration grid. In the configuration grid a cell is said to be free if and only if all of the occupancy grid points within a pre-defined distance, d, are also free. A cell is occupied if there are one or more cells within distance d that are occupied in the occupancy grid and again unknown otherwise. Finally, the controller computes a reachability grid in which cells are either *reachable* or *unreachable*. A cell is reachable if there is a chain of free cells in the configuration grid from a given cell to the location of a deployed sensor and unreachable otherwise. Once the reachability grid is computed the controller determines the goal location where the next sensor is to be deployed. Fig. 6 shows an example of the occupancy, configuration and reachability grids generated by a single node [21]. In this case, the white cells are considered to be free, the black cells are occupied and the grey cells are unknown.

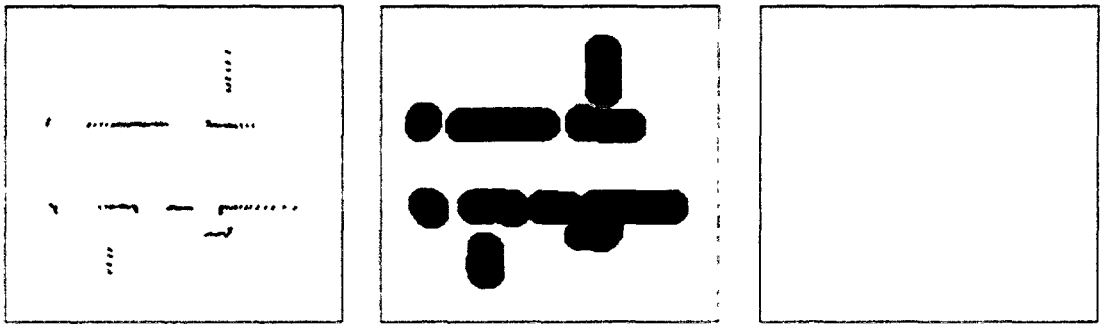


Figure 6. Sensor Self-Deployment Incremental Approach

In the assignment phase the controller assigns an undeployed to the selected goal location. The controller then computes the shortest path from the sensor to the goal location using Dijkstra's algorithm [9] through previously deployed sensors. To solve the problem of nodes blocking one another, as well as to conserve energy, movement is carried out in a shifted manner with previously deployed sensors.

In the execution phase, deployed sensors have all reached their goal locations and have begun sensing activity, gathering information about the environment. This information is then sent to the central controller which will be used to re-compute the occupancy grid when the next iteration begins. While the algorithm is presented as a sensor self-deployment algorithm, it could be applied as a carrier-based deployment algorithm, with the controller simply telling the robot(s) where to place the next sensor. However, due to the centralized nature of the algorithm, it is very expensive in terms of bandwidth and energy consumption for communication.

2.3 Coverage Maintenance

2.3.1 Cluster-Based Approach

Mei et al. presented three algorithms aimed at solving the problem of sensor replacement by mobile robots to repair coverage in a WSN [33]. The authors presented a centralized algorithm, a distributed protocol and a dynamic protocol. We will explore each in more detail below.

In the centralized protocol, a robot is chosen to be the central manager and handles all of the sensor failure reports. The central manager is stationary and the authors suggest placing it in the centre of the environment to balance message cost in all directions. Mobile robots carrying spare sensors move throughout the environment acting as network maintainers, constantly sending messages to the manager updating their current position. Sensors keep track of neighbourhood information by sending and listening to beacon messages. Sensors report node failures to the central manager. Upon receiving a failure message the manager dispatches the closest robot to the failure location. When the robot reaches the failure location it drops a spare sensor at the

location. In the event that a robot receives multiple orders, it processes the requests on a first-come, first-serve basis.

In the distributed protocol, the environment is partitioned into equal sized sub-regions or clusters. Each robot is assigned a single sub-region and is required to handle all failure reports as the central manager for the given sub-region. Each robot is also responsible for sensor replacement within its sub-region. The centralized algorithm is then run in each sub-region.

In the proposed dynamic protocol, the sensory field is dynamically partitioned according to the current location of each robot. Each robot broadcasts its current location; sensors receiving multiple messages re-broadcast only the closest location. Using this information a Voronoi diagram [40] is constructed. Sensors report node failures to the robot of their Voronoi cell. The robot will then move to replace the failed sensor, updating its current position and thus the Voronoi diagram along the way.

It is clear that all three of the proposed protocols rely on network flooding and thus are very expensive in terms of energy consumption, due to the high message cost. The authors point out that the centralized protocol can cause a communication bottleneck at the central controller in large networks where the distance the failure requests and replacement requests have to travel becomes too long. As a result, the protocols do not seem to scale well to larger networks and due to the high message and energy cost, the protocols are not well suited for practical applications.

2.3.2 Ant Colony Approach

Falcon et al. modeled the problem of carrier-based sensor relocation as a novel combinatorial optimization problem, termed *one commodity travelling salesman problem with selective pickup and delivery* (1-TSP-SELPD) [11]. The proposed problem, 1-TSP-SELPD is a generalization of the one *commodity pickup and delivery travelling salesman problem* (1-PDTSP) [19]. The problem is the following. Starting from a base station, we would like to find the shortest tour such that all delivery demands are met – that is, all sensing holes are repaired – and only profitable pick up spots are visited, before ultimately returning to the base station. Profitable here refers to minimal travel distance. The tour should not include any duplicate nodes (aside from the base station as the first

and last point) and should not allow the robot to carry more than a pre-defined limited number of sensors at any point in the tour. A sample tour is shown Fig. 7 [11], with the blue square representing the base station, the green circles representing passive sensors and the red circles representing sensing holes. In this scenario the robot can hold a maximum of three sensors at a given time and has no sensors in hand initially.

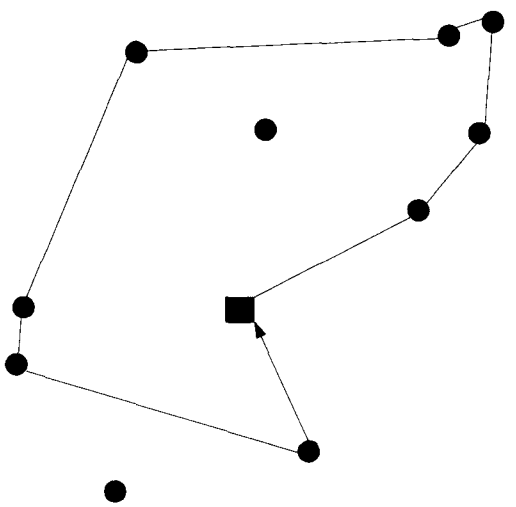


Figure 7. Sample Tour for Carrier-Based Coverage Repair

To solve the above stated problem, the authors propose to use the *ant colony optimization* (ACO) [10] meta-heuristic. At initialization, a number of ants are placed randomly at various graph nodes, with the restriction that the position must be feasible given the ant's initial cargo. The ants move concurrently and construct candidate tours, following a set of six heuristic rules. The goal of the heuristic rules is to limit the number of unnecessary movements, thus ensuring a near optimal path can be found. Once all candidate tours have completed, the shortest feasible path is determined by the base station. This tour is then reported to the robot, which then begins the tour with the required initial cargo.

This algorithm attempts to minimize the total length in the robot's trajectory as it moves to relocate redundant sensors to reported sensing holes. Due to the centralized nature of the algorithm and the frequent updates necessary to allow for the base station to have an accurate view of the number of redundant sensors, as well as sensing holes it is very expensive in terms of message cost and bandwidth. However, this was the first

proposed solution in literature to the carrier-based sensor relocation problem. As the solution proposed by Mei et al. [33] assumed robots were already carrying sensors as payload, rather than attempting to relocate redundant sensors.

2.4 Sensor Relocation

2.4.1 Mesh-Based Approach

Li et al. introduced a novel localized structure, *information mesh*, for updating and retrieving location information. Based on this structure, the authors proposed a Mesh-based Sensor Relocation Protocol (MSRP) [30]. In MSRP, every redundant sensor (*R-node*) selects its nearest active (*A-node*) neighbour to act as a proxy. Proxy nodes dynamically construct an information mesh following the iMesh protocol [31].

When a proxy node fails, its R-node moves to replace the proxy directly. When a non-proxy A-node fails, its East most, West most, North most and South most neighbours send a search message in their respective directions to find other nearby proxies using the information mesh. Once all four neighbours have received replies, the results are compared and the nearest R-node is chosen as the replacement node. The neighbour closest to the R-node proxy is referred to as the *replacement discoverer*.

To begin the node replacement process, the replacement discoverer sends a migration message to the proxy, which sends a reply back to the discoverer granting the request. Upon receiving the response the discoverer begins the shifted migration process by sending an action request to the proxy. The action request is forwarded using Greedy-Face-Greedy (GFG) routing [3] – see section 2.4 – coupled with the cost over progress ratio [39] for the greedy movement. After sending the action message the discoverer moves immediately to the position of the failed A-node. Intermediate A-nodes forward the action message, and then move to the location of the prior hop of the action message. Finally, once the message reaches the proxy node, it informs the R-node to move to its current position before moving to the location of the prior hop.

2.4.2 Quorum-Based Approach

Wang et al presented a grid-quorum-based sensor relocation protocol [41] in which the sensor field is divided evenly into grids. For each grid, one node is chosen to

act as the grid head. The grid head is responsible for collecting information from all of its grid members. Using the location information, grid heads are able to determine the existence of redundant sensors within their grid, as well as sensing holes. The authors separate the problems of finding redundant sensors and sensor relocation. We will examine their proposed solutions in more detail below.

To solve the problem of identifying sensing holes and finding redundant sensors, the authors propose to form a supply quorum and a demand quorum. The grid heads in a row make up the supply quorum, while the grid heads in a column make up the demand quorum. As previously stated, grid heads learn of sensing holes and redundant sensors within their grid using local information. When a redundant sensor is found, the grid head broadcasts this information along the supply quorum which it belongs to. When a sensing hole is detected the grid head sends a search message along its demand quorum to find the nearest redundant sensor. In order to reduce message complexity, an optimization was added; the location information of an already discovered closest redundant sensor is piggybacked on the search message in order to restrict the distance that the message travels.

For sensor relocation, the authors propose to use cascaded movement. The reasoning for this decision, is that if a sensor is required to move the entire distance on its own, not only would it require too much energy but there is a possibility that the sensor may not arrive fast enough if there is a constraint on relocation delay. As a result the redundant sensor must find intermediate nodes along the path to the destination to act as cascading nodes. The path from the redundant node to the destination is chosen by first using a flooding process within the area covering both the hole and the redundant sensor to find all intermediate sensors. Then Dijkstra's algorithm [9] is used to compute the shortest path to the destination.

Li and Santoro presented a zone-based sensor relocation algorithm, ZONER [29]. Similar to the above grid-quorum protocol presented by Wang et al [41], redundant sensors *register* themselves with the active nodes within a vertical zone – North to South – referred to as the *registration zone*. When a node failure is detected, neighbouring sensors will send a search message along a horizontal zone – East to West – referred to as

the *request zone*. The authors use zone flooding for both node registration and node request. Zone flooding is a combination of face routing and range-restricted flooding.

When a redundant sensor is found it will be relocated in a shifted or cascading manner. The path used for migration is an aggregation of the path used for registration from the replacement node to the destination and the request path from the node which discovered the nearest redundant sensor to the destination. Although similar in many ways, ZONER guarantees node registration and discovery even in the presence of void areas due to obstacles or large sensing holes by resorting to face routing. Additionally, ZONER does not require any pre-knowledge of the environment.

2.5 Message Routing

Bose et al. presented a routing algorithm with guaranteed delivery, *Greedy-Face-Greedy* (GFG) [3]. GFG first requires a connected planar subgraph to be found. The authors prove that this can be obtained by finding the Gabriel graph [14]. Given a graph with the set of vertices S, $G(S)$, then the Gabriel graph, GG(S), contains the edge (u, v) if and only if disk(u, v) contains no other points. In Fig. 8, the edge (u, v) will not be included in GG(S) because node w is in disk(u, v). However, edges (u, w) and (w, v) will be included, as no additional points lie within either of the respective disks.

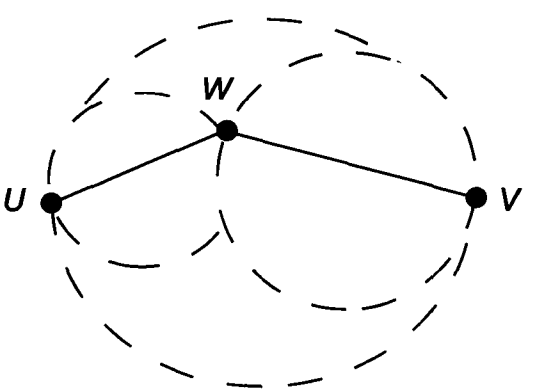


Figure 8. Gabriel Graph Example

When routing using GFG, a node greedily forwards the message to its neighbour which is closest in distance to the destination node. If no such neighbour exists, a

recovery phase is started. In the recovery phase, Face routing is used following either the right hand rule (or left hand rule) until greedy forwarding can continue (i.e. once a node is reached which is closer to the destination than the node which started the recovery)

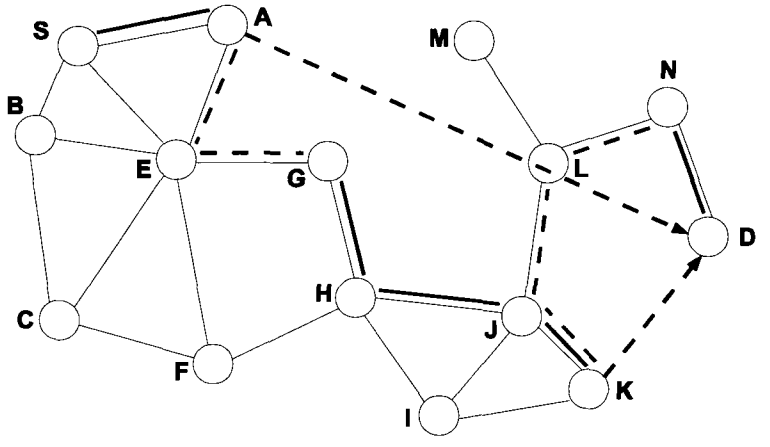


Figure 9. GFG Example

An example of GFG routing can be seen in Fig. 9, with a message is being forwarded from node *S* to node *D*. Thick red lines represent Greedy routing, blue dashed lines represent Face routing and black dashed arrows represent the node where the recovery phase is started and where the destination node is. In this scenario, a recovery phase is initiated at node *A* and again at node *K*.

Chapter 3: Back-Tracking Deployment Algorithm

As discussed earlier in Sec. 1.3, no satisfactory localized solutions to the problem of carrier-based sensor deployment exist. We showed that SLD [6, 7] cannot guarantee full coverage with a very simple scenario, containing only a single obstacle. We will show in Chapter 5 that LRV [2] is not a practical solution, as it requires a significant number of wasted movements and messages. Further, it does not support multiple robots.

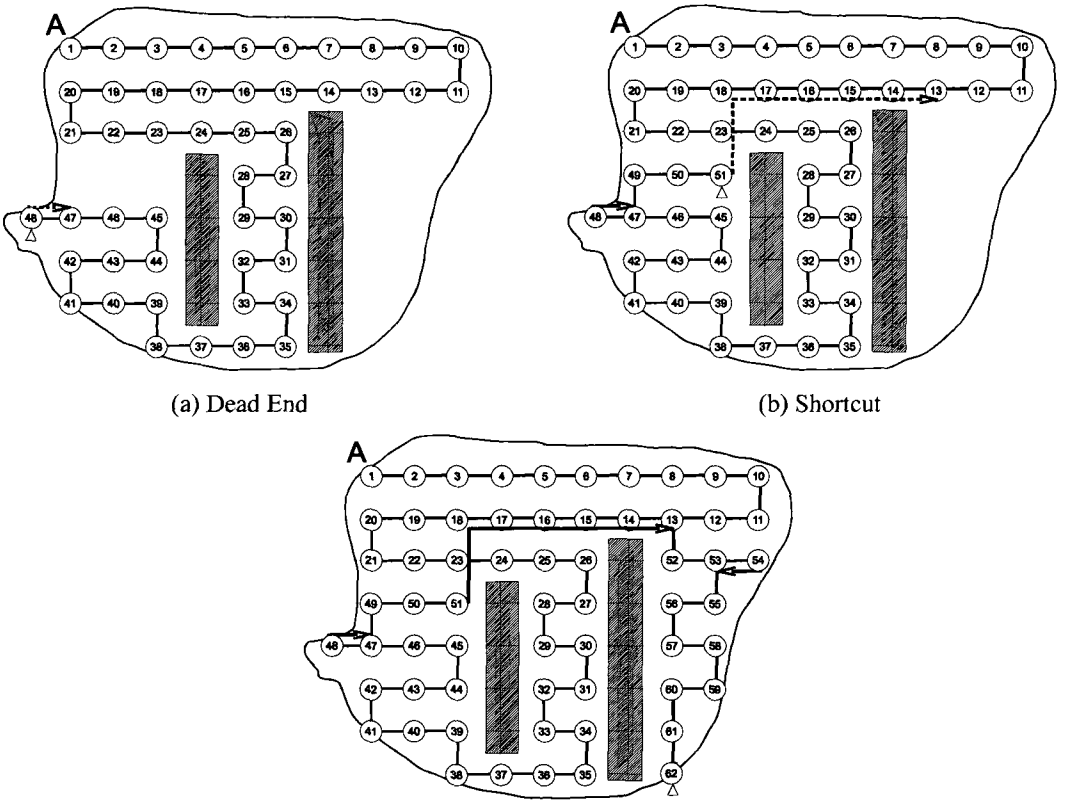
Back-Tracking Deployment (BTD) is a localized carrier-based sensor deployment algorithm. In this chapter we will describe the algorithm in detail. For ease of understanding we will first describe BTD in a failure-free environment with only one robot. We will then present the algorithm in full, removing the unrealistic assumption that sensors cannot fail and allowing for multiple robots to work as a team to deploy sensors.

3.1 Failure-Free Environment

In this section, we assume that no sensor failures may occur, and under this assumption we present the *Back-Tracking Deployment* algorithm (BTD) and prove its correctness including its claim that it terminates in finite time and its full coverage guarantee. Later, in Sec. 3.2, we will remove this temporary assumption and present a fault tolerant approach.

In BTD, the four geographic directions are pre-ordered as West, East, North, and South. This order defines the order of preference when a robot selects its movement direction. Using the given orientation, say North, robots locally compute a unique virtual square grid of edge length $\sqrt{2}r_s$ containing position (0, 0) as its grid point. An *empty point* is a grid point that is not occupied by any sensor. All grid points are initially empty. Robots collectively explore the ROI along the grid by snake-like forward moving and intelligent back-tracking. They drop a sensor at each visited empty point and inform the sensor about its adjacency to obstacles or ROI boundaries in each of the four directions. The sensor placed at the grid point where a robot is currently located is the *current sensor* of the robot. Each deployed sensor periodically transmits a "HELLO" message carrying its position and other information necessary for the deployment

algorithm. The "HELLO" message is a basic networking tool for neighbourhood discovery [23] and has been built in various networking protocols, e.g., routing protocols [3] in wireless ad hoc networks. By listening to "HELLO" messages, sensors know about the position of neighbours and any neighbourhood change (e.g., node failure and node status change). By listening to "HELLO" messages, robots are able to detect previously deployed sensors.



(a) Dead End

(b) Shortcut

(c) Full Robot Trajectory

Figure 10. BTD With a Single Robot

For forward moving, each robot proceeds step by step to grid points in *open* directions, based on the directional order of preference, until a *dead end* is reached. A direction is closed if the robot's movement step to that direction is obstructed by an obstacle, an ROI boundary or a previously deployed sensor, and open otherwise. A dead end is defined as a grid point in which all four directions are obstructed; thus the robot can no longer move forward. In Fig. 10(a), the robot's current location (marked by a

small triangle) is a dead end. In a dead end situation, the robot will back track to the nearest sensor adjacent to an empty point along the backward path of a robot (with preference to its own path), and resume exploration of the ROI and sensor dropping from there, or stop moving if no such a sensor exists.

At the core of BTD is indeed robot back-tracking. In the following, we will elaborate how a robot performs back-tracking merely using local information. For ease of understanding, we will first consider the single-robot scenario and then examine the more complex situation where multiple robots are present.

3.1.1 Single Robot Scenario

Each deployed sensor stores three pieces of information, *sequence number*, *color*, and *back pointer*. A sensor colors itself *white* if it is adjacent to an empty point and *black* otherwise. It updates its own color dynamically. This means that a white sensor may become black, as the robot continues to place sensors throughout the ROI. The sequence number is a monotonically increasing number assigned by the robot. It indicates the order in which the sensors were placed. By exchanging sequence number – through "HELLO" messages – sensors are aware of the successor/predecessor along the robot's path amongst its neighbours. The back pointer points to the location of the first white sensor along the robot's backward path. It is established dynamically in a cascading way via "HELLO" message, from the successor of a white sensor to the next white sensor along the forward path of the robot. When the robot needs to perform back-tracking, it can find the destination by looking at the back pointer stored on its current sensor.

Fig. 10 illustrates a robot's trajectory following the BTD algorithm. In this scenario, the robot starts at position *A* and its current position is marked by a small triangle. The robot's trajectory is marked by thick, solid lines. Arrowed lines indicate the shortcut path taken by the robot during back-tracking. Sensors are represented by circles, with their sequence number enclosed.

In Fig. 10(a), node 13 includes its color and back pointer (to node 12) in a "HELLO" message. After node 14 receives the message, it sets up its back pointer to node 13 and includes the back pointer and its own color in its own "HELLO" message. Then node 15 hears this information, sets its own back pointer to node 13 because node

14 is black, and forwards its information. Finally, nodes 1−11 have no back pointer; 12 and 13 have back pointers pointing to 11 and 12, respectively; nodes 14 − 21 have the same back pointer, pointing to node 13. In this snapshot the robot has reached a dead end. The robot moves to the location of the back-pointer at its current sensor, node 47 in this case and can continue exploration. In Fig. 10(b) the robot has once again reached a dead end and will back-track to node 47. Finally, in Fig. 10(c) the robot terminates movement since no back pointers can be found.

Three different back-tracking movement methods may be used. The first method consists of the robot retracing its path until reaching its destination. It will require the robot to move in a zigzag fashion, resulting in an unnecessarily long moving distance. The second method requires the robot to move directly, along a straight line, to its back-tracking destination. This method is vulnerable to obstacles that prevent the robot from taking the most direct path. In this case without an extra obstacle avoidance algorithm the robot could get stuck by the obstacles. Even with an obstacle avoidance algorithm such as face routing, the robot could take an inefficient path to reach its destination due to the lack of global view.

The third back-tracking method is an optimized version of the first method. It is optimized in terms of minimizing (based on local knowledge) the number of moves that is required for the robot to reach its destination. The robot moves to the next adjacent sensor with the lowest sequence number whose back pointer location is the same as the robot's destination. This *shortcut* method is illustrated in Fig. 10(b). The dashed arrowed lines represent the path that robot will take in order to reach the destination as specified by the back pointer at its current location using the shortcut, jumping over intermediate sensors along the original backward path.

The shortcut method achieves two goals. It allows the robot to reach its destination regardless of obstacles, as the retracing method does. It is also a more efficient method, eliminating wasted movement by the robot. For these reasons the shortcut method is employed by BTD for the back-tracking portion of the algorithm. Fig. 10(c) shows the full robot trajectory with thick solid lines indicating the robots forward path and thick arrowed lines indicating the back-tracking paths taken by the robot.

The BTD algorithm pseudo code for the robot and for sensors is shown in Alg. 1 and 2, respectively. In Alg. 2, previousSensor for a sensor, i, refers to the sensor with sequence number $i-1$.

Algorithm 1. Back-Tracking Deployment (BTD), robot movement

```
1:  while back pointers exist
2:      if no sensor within communication range then
3:          place sensor
4:      else if West isEmpty then
5:          move West
6:      else if East isEmpty then
7:          move East
8:      else if North isEmpty then
9:          move North
10:     else if South isEmpty then
11:         move South
12:     else
13:         while currentLocation ≠ back pointer
14:             Perform backtracking
15:         loop
16:     end if
17: loop
```

Algorithm 2. Back-Tracking Deployment (BTD), sensor

```
1:  if adjacent to empty space then
2:      colour := white
3:  else
4:      colour := black
5:  end if
6:  if previousSensor.colour = white then
7:      backPointer := previousSensor
8:  else if previousSensor.backPointer ≠ null then
9:      backPointer := previousSensor.backPointer
10: else
11:     backPointer := null
12: end if
```

3.1.2 Multi-Robot Scenario

When multiple robots are present and placing sensors, each sensor needs to store an additional piece of information, *robot ID*, which indicates which robot it was dropped by. It should be noted that each robot independently maintains its sequence number which is then passed on to the sensors that it places. Therefore, it is possible for two sensors to share the same sequence number if they were placed by different robots;

however, no two sensors placed by the same robot may share the same sequence number. Thus, sensors can be distinguished by the value pair (*robot ID*, *sequence number*).

Each robot follows the BTD algorithm as if it was the only robot in the ROI. It informs its placed sensors of its ID. In a dead-end situation, if it cannot find a back pointer on its current sensor, it will randomly select a back pointer (if any exists) with the largest sequence number stored in the neighbourhood and back track to the indicated white sensor. Due to network asynchrony and communication delay it is possible for a robot to move to a black sensor from a dead end. In this case, the robot will have reached another dead end.

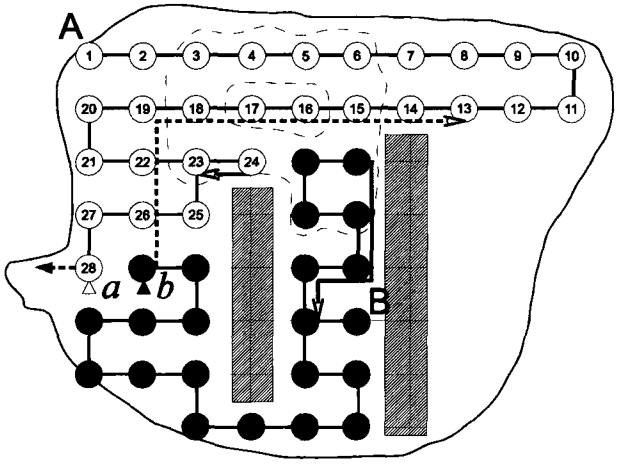


Figure 11. BTD With Multiple Robots

When a robot is back tracking to a white sensor, other robots should not follow the same back-track path for that white sensor. Therefore during back tracking, the robot informs its current sensor to erase back pointers along the forward path to the first encountered white sensor or the first sensor that stores no back pointer. This can be implemented using a "HELLO" message in order to reduce communication overhead. For example, in Fig. 11, robot *b* decides to back track along robot *a*'s path to node (*a,* 13). During the back tracking, after it reaches node (*a,* 26), it informs the node to erase the back pointer. Node (*a,* 26) then does so and includes information in its "HELLO" message for the successor to erase its back pointer. After receiving the message, the successor node – (a, 27) – erases the back pointer from its memory and forwards the erase

request in its own "HELLO" message. In this way, robot *a* will not perform back tracking or its back tracking will be stopped due to the deletion of the back pointer.

3.1.3 Load Balancing

Once a robot has reached a dead end with no back pointer in its neighbourhood, it will no longer terminate. Instead, the robot will send a message searching for a nearby white sensor in an arbitrary direction – North, for example. The message will be routed using the GFG principle. In the event that no white sensor or back pointer is found, the message will traverse the network and reach the Northern-most node. In the event of a tie, North-East is considered more Northern. At this point the Northern-most node will send a reply to the robot indicating that no back pointer was found. Upon receiving this reply, the robot will terminate.

If during the message routing, a white sensor or a sensor which has a back pointer is encountered, a reply is sent back to the robot indicating the position of the white sensor and the message routing is stopped. By sending the reply from the first encountered white sensor, the "busy wait" time for the robot is minimized. Additionally, given that it is the first encountered white sensor along the message path, the distance between the robot and the white sensor will be minimized. Once the robot received the reply message, it will move to the target destination and continue exploration.

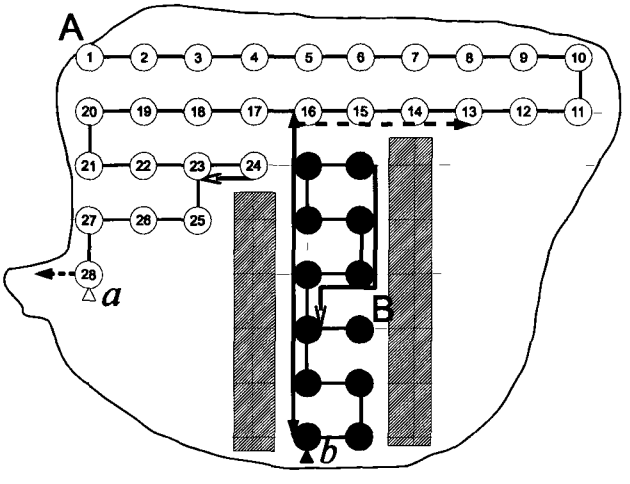


Figure 12. Load Balancing in BTD

An example of the load balancing technique can be seen in Fig. 12. Two robots, *a* and *b* start from positions *A* and *B*, respectively. In this scenario, robot *b* has reached a dead end at node (*b*, 12), with no back pointers in its neighbourhood. Robot *b* sends a message to the North attempting to find a white sensor within the ROI. The message – represented by the thick two-sided arrow – eventually reaches node (*a*, 16), which replies with the location of its back pointer – represented by the dashed arrow – in this case node (*a*, 13). Robot *b* will then move to node (*a*, 13) and continue node dropping in the unexplored area. It is clear even in this simple scenario that without this load balancing technique, robot b would have only placed 12 sensors, while robot a would have been tasked with placing the remainder.

By actively searching for back pointers within the ROI and continuing exploration, load balancing is improved in BTD. This also helps to reduce the energy consumed per robot. Additionally, because all of the robots remain active, as in Fig. 12, the time to fully explore the ROI is also reduced.

3.1.4 Proof of Correctness

Lemma 1: Each grid point can be visited at most 7 times in total and at most 4 times by the same robot.

Proof: Every time a robot reaches a grid point, the robot visit count at the point increases by 1. Any point with a sensor node has robot visit count at least 1. A robot always visits an empty point unless it is performing back-tracking. Let us assume that there is no information propagation delay. Consider a grid point where a black sensor is located. When a robot visits the sensor (point) during back-tracking, the back pointer stored on that sensor is erased. No other robot will visit it afterwards. Because back tracking goes through sensors with monotonically decreasing sequence number, a robot will not visit the same black sensor for back-tracking more than once.

Consider a point where a white sensor is located. If the point is the starting point of a robot, the white sensor will have at most three open directions before it becomes black. Therefore, it will be visited at most 3 times as a back-tracking destination. After these visits, it becomes black, and there will be no more back-tracking through it because it has the smallest sequence number. If the point is not a robot starting point, the white

sensor will have at most two open directions, and possibly a back pointer to another direction. A back-tracking destination can be visited at most twice before it turns black, and at most one more time after as intermediate node of back-tracking.

Summarizing, any point will be visited at most 4 times, 1 during robot forward moving and 3 during robot back tracking. These visits are effective for sensor placement. Trivially, we can conclude that each robot will visit the same point no more than 4 times. This conclusion clearly does not depend on the assumption of zero information propagation delay.

In the presence of information propagation delay and multiple robots, ineffective (mistaken) robot visits are possible. A point can be visited by a robot from 4 directions. No different robots will visit the same point from the same direction regardless of the purpose of the visit. Thus, the point will be visited at most by 4 different robots, and 3 of these robot visits will be ineffective. Hence, each point will be visited at most 7 times overall. ■

Theorem 1: BTD terminates within finite time.

Proof: BTD terminates once all robots stop moving permanently. From Lemma 1, a robot can visit a sensor no more than 4 times. The number of sensors, n, is bounded and equal to the number of grid points contained in the ROI. Hence, the maximum number of movements that a robot can perform is $4n$, implying each robot will make a finite number of moves between grid points. ■

Theorem 2: BTD yields full coverage over the ROI.

Proof: Assume for the sake of contradiction that, after the algorithm terminates, there is an empty point in the ROI. The sensors adjacent to this empty point are coloured white. Then back pointers pointing to these white sensors are established along the forward paths of their placing robots. According to the algorithm, these robots will eventually back track to the white sensors unless they never reach a dead end (i.e., able to keep moving forward forever), contradicting in either case, our assumption that the algorithm has terminated. ■

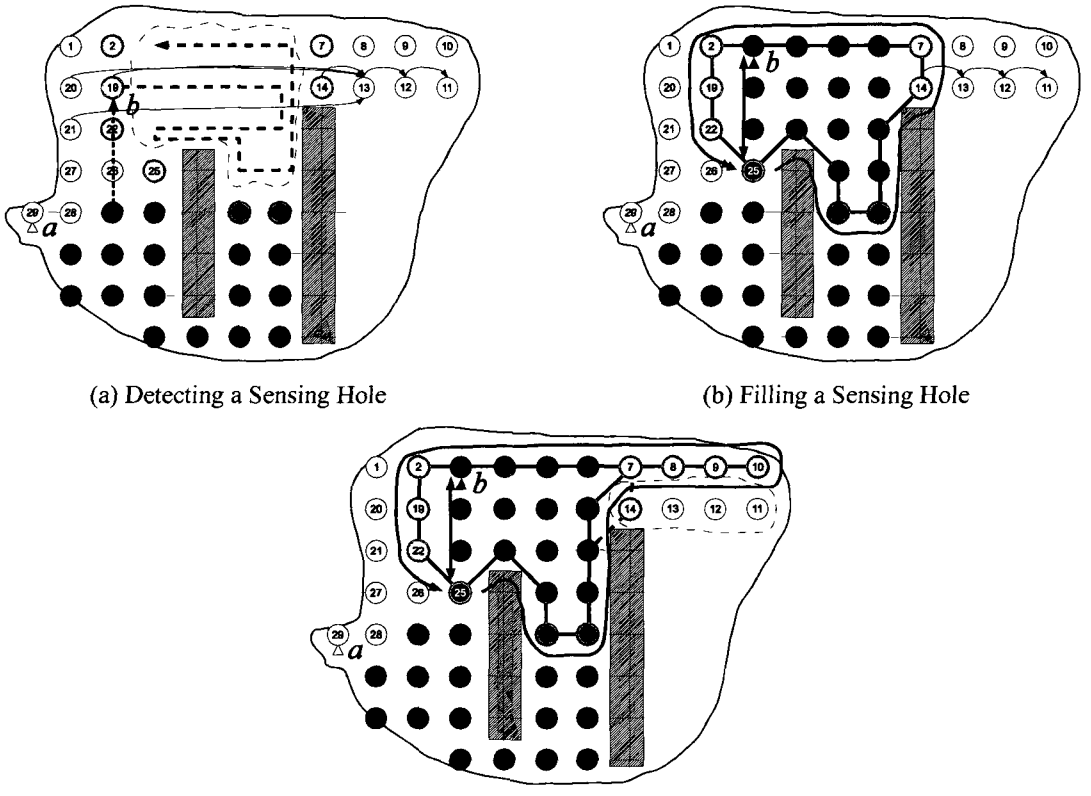
3.2 Failure-Prone Environment

In this section we consider a failure-prone environment where sensing holes are present as a result of sensor failures. The presence of sensing holes has the following two negative impacts on the sensor placement process. Robots use locally identified shortcut paths for back tracking; if a back-tracking shortcut passes through a sensing hole, the robot will fail to follow it. The back pointer chain leads the robot from one white sensor to another; sensing holes may break the chain and prevent robots from fully exploring the ROI.

In this section we will first describe our strategy for recovering from *acyclic sensing holes* and in Sec. 3.2.3 we will describe the recovery from *cyclic sensing holes*. We term continuous sensing holes (see Fig. 13(a)) acyclic and cyclic otherwise. In Fig. 11, if only the sensors located between the two dashed circles fail, a cyclic sensing hole would emerge. Recovery from cyclic sensing holes is more difficult, as there are both inner and outer boundary nodes that the robot must consult as it attempts to recover.

3.2.1 Finding the Recovery Agent

To handle the sensing hole problem, we introduce a new color, *gray*. Sensors adjacent to a sensing hole (i.e., failed sensors) colour themselves gray. The sensing hole is bounded by the gray sensors, the adjacent obstacles and the ROI boundaries, as illustrated in Fig. 13(a) where gray nodes are: (*a,* 2), (*a,* 7), (*a,* 14), (*a,* 19), (*a,* 22), (*a,* 25), (*b,* 3) and (*b,* 4). A robot is able to identify a sensing hole locally as soon as it finds that its back-tracking shortcut is broken due to loss of track of the back pointer and its current sensor is a gray sensor.



(a) Detecting a Sensing Hole

(b) Filling a Sensing Hole

(c) Tolerating Gray Sensor Failure

Figure 13. Fault Tolerance

In Fig. 13 sensing holes are represented by dashed circles surrounding one or more sensors. If a sensor has a back-pointer, it is represented by a thin arrowed line. Dashed arrowed lines indicate the robot's trajectory within the sensing hole in order to fill it. Thick arrowed lines indicate the message transmission path, while thick non-arrowed lines highlight the ring network surrounding the sensing hole. Finally, the black node is the *recovery agent.*

If all the failed neighbouring sensors of the current gray sensor have been replaced, the robot considers the hole filled and will try to recover the back-tracking shortcut via the current gray sensor, referred to as the *recovery agent* in this context. If the hole is not yet filled, the robot will treat it as a new bounded unexplored area and run the BTD algorithm recursively to fill it.

For instance, in Fig. 13(a) robot *b*'s back-tracking (along robot *a*'s path) is interrupted at node (*a,* 19) by the sensing hole. At this moment, the back pointers along

the forward path from that node to the back-tracking previous hop (*a,* 22) has not been erased. The robot informs (*a,* 19) to delete back pointers along the forward path, enters the hole, and runs BTD along the trajectory shown as dashed arrowed line.

After finishing the algorithm inside the hole, the robot sends a message by Greedy-Face-Greedy (GFG) routing [3] in an arbitrary direction – South, for example – as shown in Fig. 13(b). The message carries the back pointer (pointing to (*a,* 13) in this example) and the identification (*a,* 19) of the sensor where back-tracking was interrupted. It will hit a gray node (*a,* 25), which is then the *recovery agent.*

Notice that after the sensing hole is completely filled, the gray sensors and the replacement sensors adjacent to the physical obstacles and the ROI boundaries constitute a *ring network*, as illustrated in Fig. 13(b). The recovery agent is a gray sensor on the ring network. On behalf of its delegated robot, it will search, along the ring, for a sensor where the robot can resume the back tracking or start a new back tracking. We will present the search process in the next sub-section.

3.2.2 Finding the Back Pointer

The recovery agent sends a search message carrying the back pointer that its delegated robot was following and the associated robot ID and sequence number along the ring. Ring traversal can be done by face routing [3]. The search message will erase the same back pointer with larger sequence number; meanwhile, it will pick the location of the sensor with largest sequence number among those that were dropped by the same robot and whose back pointer remains, as well as the corresponding back pointer (which may be different from the initial target back pointer) and sequence number information. After the message makes a full circle, it carries the search result back to the recovery agent. Timeout-based search retrial may be needed until the ring network is fully formed and the ring traversal succeeds.

If the search succeeds, the recovery agent will find by another ring traversal the sensor that has the same back pointer and smallest sequence number. The intuition is the same as that of finding the shortcut. If the search fails, the recovery agent will also initiate another round of search. In this round, it does not specify any preferred robot, and the search message picks the back pointer it discovers first and then looks for the

sensor with the same back pointer and smallest sequence number. The recovery agent forwards the final search result to the robot. This search process at the recovery agent is shown in Alg. 3.

If the search result is positive, the robot will relocate itself by GFG principle [3] to the specified sensor location. Otherwise, it will move to the original back-tracking destination by Greedy-Rotation-Greedy (GRG) principle [24, 25]. The objective is to tolerate gray node failure. For example, in Fig. 13(c), if the sensors in the dashed circle fail before the back pointer search is completed and after the hole is filled, it is possible no back pointer can be discovered. In this case, if robot *b* does not move to (*a,* 13), then the ROI exploration fails and so does sensor placement.

After reaching the relocation destination, the robot performs back-tracking following the discovered back pointer. During the relocation to the destination, the robot may meet another sensing hole. For example, in Fig. 13(c), robot *b* will meet the new sensing hole when moving to (*a,* 13). In this case, it will repeat the above fault-tolerance algorithm.

Algorithm 3. Back-Tracking Deployment (BTD), recovery agent

```
1: send search message with preferred robot information to find back pointer
2: if search fails then
3:     send search message without preferred robot information
4: end if
5: send search message with discovered back pointer to find sensor with the smallest sequence
   number
6: return back pointer and the location of the sensor with the smallest sequence number
   containing the back pointer
```

3.2.3 Recovery From Cyclic Sensing Holes

To deal with the problem of cyclic sensing holes, after filling a sensing hole, the robot will send a search message in all four directions, eventually traversing the hole boundary. It is possible that the search will return multiple back pointers, due to replies coming from inner boundaries, as well as outer boundaries. The robot will choose the back pointer with the smallest sequence number and remember all others.

If after to moving to the destination, the robot finds that the sensor is no longer white – it is no longer adjacent to an uncovered area and thus no longer valid – it puts the

back pointer on a black list and moves to the back pointer with the next smallest sequence number (if any). If the entire list of back pointers is exhausted with no success of discovering an uncovered area, the robot will begin a new search process, embedding the black listed back pointers in the search message this time. The search message will delete the outdated back pointers from the nodes that it visits. The search message will return a new set of back pointers. The above process is repeated until the robot discovers a successful back pointer or until all back pointers have been deleted, at which point the ROI has been fully explored.

Chapter 4: Randomized Robot-Assisted Relocation of Static Sensors Algorithm

Recall, from Sec. 1.3, that there are currently no localized solutions for the carrier-based sensor relocation problem. We will thus present the first solution, Randomized Robot-Assisted Relocation of Static Sensors (R3S2). In R3S2, robots move randomly throughout the ROI picking up relocating reported redundant sensors to locally identified sensing holes. We attempt to limit the robot's randomness by introducing a grid-based variant. We then introduce a cluster-based variant to fill holes more efficiently.

Due to the localized nature of the algorithm, an activity scheduling algorithm [15] is adopted for determining passive sensors. By this algorithm, sensors determine their own status – passive or active – using one-hop neighbourhood information. For energy-saving, passive sensors turn off their radio module, falling "asleep". Before doing so, they choose a closest active neighbour to act as a *proxy* by sending a delegation message to it. Nodal relative position can be used as a tie breaker for proxy selection.

4.1 Basic Techniques

In this section we will briefly review basic techniques that we will use in the cluster-based variant, extending the basic algorithm. We will first review connected dominating sets (CDS) and a localized algorithm which can be used to build the CDS. We will then review a virtual force algorithm and describe how it can be used for sensor relocation to fill sensing holes.

4.1.1 Connected Dominating Sets

A dominating set for a graph $G = (V, E)$ is a set of vertices $D \subseteq V$ such that every vertex not in D is a neighbour of at least one vertex in D. A connected dominating set (CDS) has the additional property that the graph induced by D is connected. That is, any two nodes in the CDS are able to communicate with each other.

The problem of finding a minimal CDS can be shown to be an NP-complete problem [18]. Dai and Wu proposed a generalized self-pruning algorithm for constructing a CDS [8]. The goal of the algorithm is to prune nodes which will not be a part of the CDS. To do this the algorithm considers a node's *priority* as a means of comparison. Priority refers to a pre-defined key, such as a node's unique ID, though more complex keys can be used; taking into account the number of neighbours or remaining battery life, for example.

The following is a brief description of the generalized self-pruning algorithm proposed by Dai and Wu. The algorithm first attempts to find all *intermediate* nodes. An intermediate node is defined as a node which has at least two unconnected neighbours. A non-intermediate node will never be in the CDS and can thus be "pruned" right away. Then for every intermediate node, the algorithm checks whether the node is *covered.* A sensor, S, is covered by a subset of its neighbours, N, if each of the following conditions hold:

- N is connected
- Any neighbour of S not in N is also a neighbour of at least one node in N
- $Key(S) < min(Key(N))$

If a node is found to be covered then it is not dominant and can be pruned from the CDS. All remaining nodes will be part of the CDS.

An example of CDS construction can be seen in Fig. 14. In this example we are given a network with 14 nodes, as seen in Fig. 14(a). We will use the node ID as the key, such that the greater ID has greater priority. Nodes are assumed to know each of their neighbours ID.

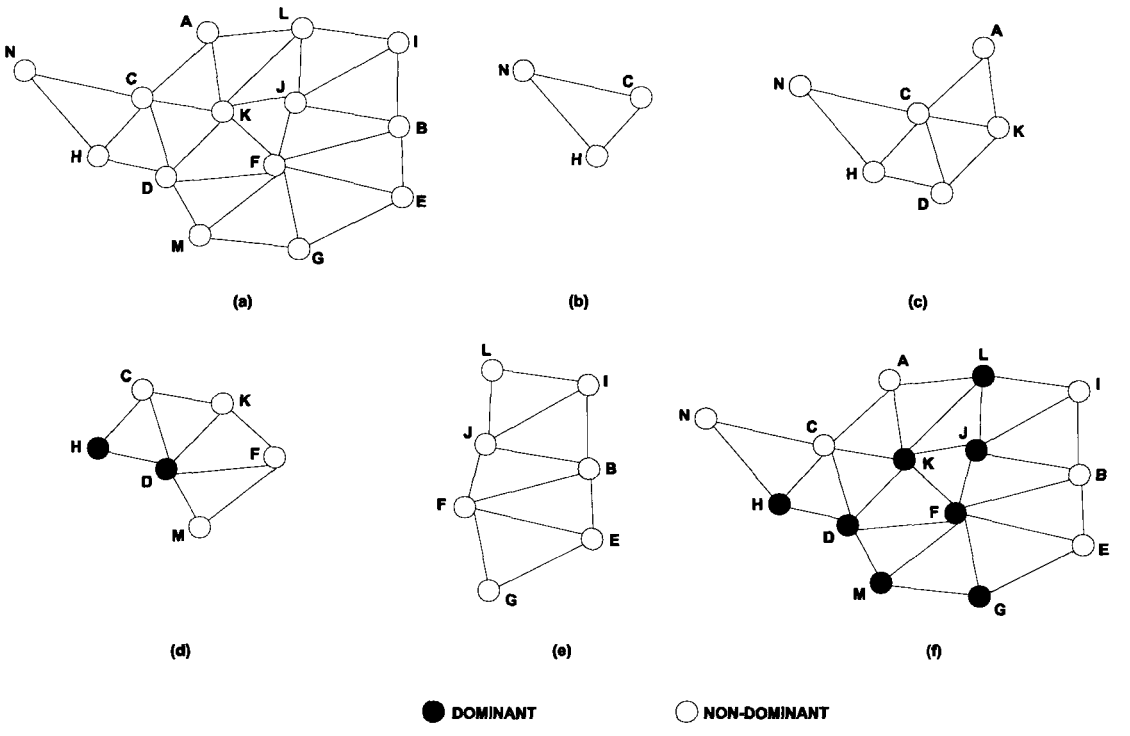


Figure 14. CDS Construction

In Fig. 14(b), node *N* is discovered to be non-intermediate, as it does not have at least two unconnected neighbours. In Fig. 14(c), node *C* is covered by its neighbours {*H, D, K*} since all three neighbours are connected, all three have a greater ID than C and its remaining neighbours, {*N, A*} are neighbours of *H* and *K*, respectively. In Fig. 14(d), node *H* is able to determine that it is dominant and thus in the CDS. *H* is able to make this decision since its only neighbour that has not been pruned is *D*, which has a smaller ID. Thus, *H* is not covered. *D* is also able to decide to become dominant, as the subset of its neighbours with greater priority, {*H, K, F, M*}, are not connected. In Fig. 14(e), nodes *I, B* and *E* are determined to be non-dominant. *E* is covered by neighbours {*F, G*}, *B* is covered by its neighbours {*F, J*} and *I* is covered by its neighbours {*J, L*}. Additionally, given that *A* has the lowest ID, it will be covered by all of its neighbours and thus be non-dominant. The remaining nodes are dominant and the resulting CDS, {D, F, G, H, J, K, L, M}, can be seen in Fig. 14(f).

The decision by a sensor whether it is dominant or not can be made locally given single-hop neighbour information and the position of all single-hop neighbours. Therefore no additional message cost is incurred for the sensor relocation algorithm.

4.1.2 Virtual Force Algorithm

Zou and Chakrabarty proposed a virtual force algorithm (VFA) for deploying mobile sensors to maximize sensing coverage [44], [45]. The algorithm simulates the forces felt by point charges. We will consider the sensors, as well as obstacles as particle points. As with electric forces, the sensors can experience both attractive and repulsive forces acting upon them. An obstacle within the ROI will exert a repulsive force on nearby sensors. The authors also mention that areas of preferential coverage (assumed to be known a priori) within the ROI will exert an attractive force on nearby sensors. Neighbouring sensors may exert an attractive force, a repulsive force or no force at all on neighbouring sensors depending on their proximity.

$$\overrightarrow{F_{ij}} = \begin{cases} \left(w_A(d_{ij} - d_{th}), \alpha_{ij}\right) & if\ d_{ij} < d_{th}, \\ 0 & if\ d_{ij} = d_{th}, \\ \left(w_R(\frac{1}{d_{ij}}), \alpha_{ij} + \pi\right) & otherwise \end{cases} \quad (1)$$

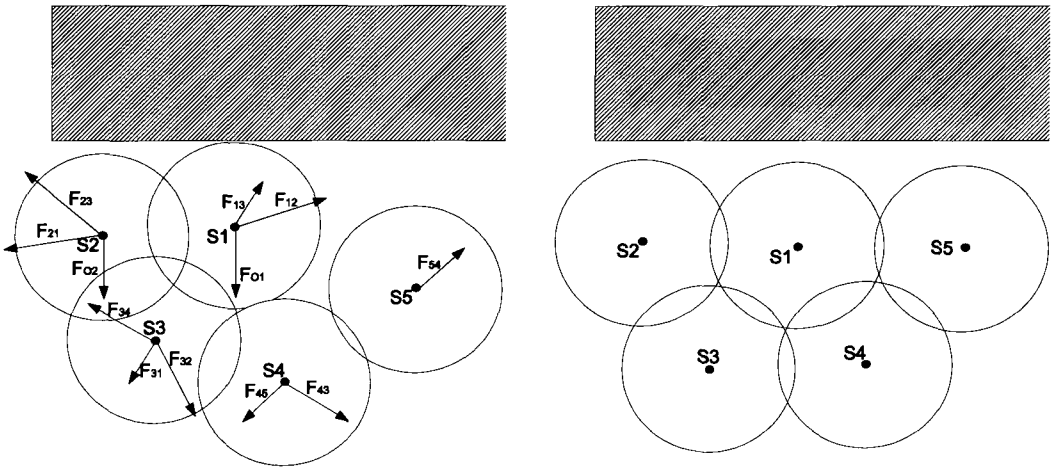
We will represent the total force acting on a sensor, s_i as $\overrightarrow{F_i}$. The force exerted on s_i by another sensor, s_j will be represented by $\overrightarrow{F_{ij}}$. Because force is a vector, it is represented by a magnitude and direction. Eq. 1 shows the calculation for $\overrightarrow{F_{ij}}$, where d_{ij} is the Euclidean distance between s_i and s_j, d_{th} is the desired nodal separation, α_{ij} is the angle of the line segment from s_i to s_j and w_A (w_R) is a measure of the attractive (repulsive) force. Recall, that the Euclidean distance (d_{ij}) between two points, say $s_i = (x_i, y_i)$ and $s_j = (x_j, y_j)$ is given by $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

$$\overrightarrow{F_i} = \sum_{j=1, j \neq i}^{k} \overrightarrow{F_{ij}} + \overrightarrow{F_{iA}} + \overrightarrow{F_{iR}} \quad (2)$$

Eq. 2 shows the vector summation for calculating the total force $\overrightarrow{F_t}$. Where $\overrightarrow{F_{tR}}$ is the total repulsive force acting on the given sensor from nearby obstacles (depicted as black blocks in Fig. 15 below), $\overrightarrow{F_{tA}}$ is the total attractive force acting on the sensor from areas of preferential coverage within the ROI and $\overrightarrow{F_{tj}}$ is calculated as above in Eq. 1.

Though not originally discussed by the authors, node oscillation can be controlled by having the cluster head remember its members' previous location. If the net force for a given sensor would return it to its previous location, the cluster head ignores this movement attempt. The algorithm would terminate once all cluster members movement has stopped (i.e. net force is 0 or oscillating movement is being ignored).

Depending on coverage criteria the virtual force algorithm can provide a final sensor configuration with either *overlapped* or *non-overlapped* coverage. If the sensing radii of neighbouring sensors overlap, we refer to the sensor configuration as having overlapped coverage and non-overlapped otherwise. Overlapped coverage ensures that all grid points are covered (i.e. no gaps between sensors); however, because the sensors are more closely deployed, it requires more sensors to be deployed than non-overlapped coverage [44], [45]. Whether the coverage will have overlaps or not and to what degree is controlled by carefully selecting the d_{th} value.



(a) Initial Sensor Distribution (b) Final Sensor Distribution

Figure 15. Sensor Relocation Using Virtual Forces

An example illustrating the virtual forces algorithm can be seen in Fig. 15. Fig. 15(a) shows the initial sensor distribution, along with the forces applied to each sensor by neighbouring sensors and physical obstacles; however, there are no areas identified as preferential coverage areas. Fig. 15(b) shows the final sensor distribution following the sensors' relocation based on the forces applied to each sensor.

4.2 Local Sensing Hole Identification

An active sensor S learns the positions of active neighbours by periodic "HELLO" messages. Through local computation it identifies the arcs on its sensing range perimeter that are not covered by any active neighbour. Any uncovered arc that is larger than, or equal to, degree $\pi/2$, will be divided evenly in half. Arc division is carried out, at most 3 times in the worst case that the initial arc has a degree of 2π, until all uncovered arcs have a degree less than $\pi/2$.

Finally, each uncovered arc implies an adjacent sensing hole. S will consider that the number $N_h(S)$ of adjacent holes is equal to the number $N_a(S)$ of uncovered arcs. In reality, $N_h(S)$ can be less than $N_a(S)$ (even without intentional arc division) as two local holes are possibly part of the same large hole. This inaccuracy is inevitable in a localized algorithm.

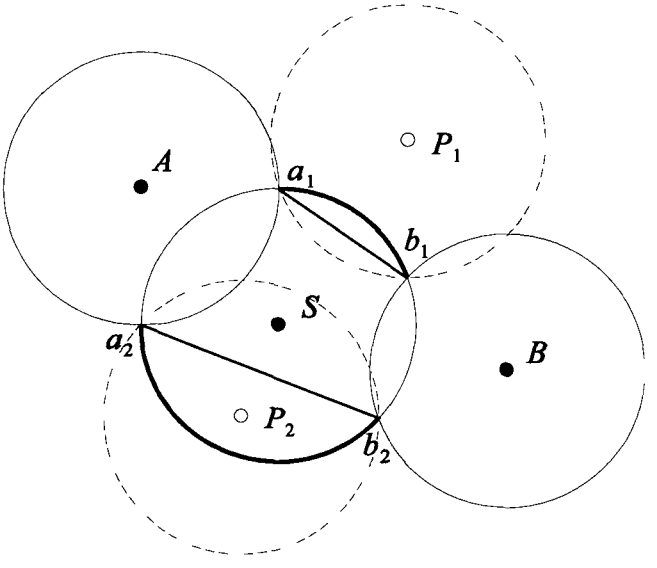


Figure 16. Local Sensing Hole Identification

For a local hole defined by an uncovered arc, its position (i.e., center) is defined as the position symmetric to the location of S by the chord of the arc. For example, in Fig.16, S has two uncovered arcs a_1b_1 and a_2b_2 (thickened) on its sensing range perimeter; it decides that there are two holes respectively at positions P_1 and P_2. The intuition is that, if an additional sensor S' is placed there, the hole will be exactly filled with minimum sensing range overlapping between S and S'.

4.3 R3S2 Algorithm

Robots travel within the ROI step by step autonomously and asynchronously at random; they do not communicate while moving. For every robot R, there is a discovery phase of pre-defined length between any two successive movement steps. In this phase, R remains static, and periodically transmits a beacon message carrying its current location. On receiving the beacon message, nearby active sensors reply with the locations of their adjacent passive sensors and sensing holes.

Let $\boldsymbol{H}$ and $\boldsymbol{V}$ respectively be the hole set and the passive sensor set discovered by robot R during a discovery phase. Denote Φ as the empty set. Denote by $\boldsymbol{S}$ the set of sensors currently held by R. At the end of the discovery phase, R makes a decision on its next movement, either continuing random movement (in this case the robot is said to be free) or moving to fulfill a task (in this case, the robot is said to be busy) – picking up a passive sensor and/or fixing a hole, according to the following policies:

1. In the case of $\boldsymbol{H} = \Phi$ and $\boldsymbol{V} = \Phi$, there is no task for R. R continues its random movement.
2. In the case of $\boldsymbol{H} \neq \Phi$ and $\boldsymbol{V} \neq \Phi$, R chooses to repair a hole $h \in \boldsymbol{H}$ with a sensor $v \in \boldsymbol{V}$, such that $|Rv| + |vh|$ is minimized.
3. In the case of $\boldsymbol{H} \neq \Phi$ and $\boldsymbol{V} = \Phi$, R chooses to repair the closest hole $h \in \boldsymbol{H}$ if $\boldsymbol{S} \neq \Phi$ or continue its random movement otherwise.
4. In the case of $\boldsymbol{H} = \Phi$ and $\boldsymbol{V} \neq \Phi$, R decides to continue its random movement with probability $1 - p$ and to pick up the closest passive sensor $v \in \boldsymbol{V}$ with probability p. Where $p = 1$ if $\boldsymbol{S} = \Phi$ or $p = 1/|\boldsymbol{S}|$ otherwise.

In Case 4, the purpose of probabilistic pickup is to increase the chance that every robot discovers a passive sensor and fixes its encountered holes. After a busy robot finishes its task, or when the task is cancelled (see below), the robot starts a new discovery phase immediately at its current location.

If two robots are neighbouring each other during their discovery phase, they will discover each other and compete for any common task. If the competition is for picking up a passive sensor (as in Case 4), the robot carrying a smaller number of passive sensors wins; if it is for a repairing hole (as in Case 3), the robot with a larger number of passive sensors wins; if it is for a pair of hole and sensor (as in Case 2), the robot with shorter travel distance wins. The winner proceeds as usual, while the loser removes relevant objects (sensors or holes) from consideration and re-makes its movement decision.

Every time a robot picks up a passive sensor, it informs the corresponding proxy to remove the sensor from further consideration. If the target sensor does not exist anymore (e.g., due to the pickup by another robot), a robot simply cancels its task. Having arrived at the target hole, a robot tries to re-discover the hole by listening to "HELLO" messages of neighbouring sensors. The robot drops a sensor if and only if the hole still exists, or cancels the task otherwise. A relocated passive sensor becomes active and broadcasts a "HELLO" message to declare its arrival. All active sensors receiving the "HELLO" message re-assess local holes.

4.4 Grid-Based R3S2

In R3S2, a robot attempts to traverse the ROI by random movement, and discover and handle all sensing holes and passive sensors encountered. The robot moves completely at random when it is free. Unlike R3S2, the grid-based version (G-R3S2) imposes some constraints on robot movement in order to lower randomness, shorten expected traversal time, and reduce coverage repair delay. However, G-R3S2 does not require any changes to the sensor deployment. Therefore a random deployment strategy can still be used.

G-R3S2 is the implementation (with improvement) of R3S2 on a virtual gird, where the movement of a free robot is restricted on the grid. Any grid such as square grid or triangular tessellation or hexagonal grid can be used. But its edge length must be

selected properly so that the sensor and hole discovery problem in the continuous ROI is translated into the traversal problem on a discrete grid. At initiation, robots spontaneously align themselves with the grid by moving to a closest grid point. They then move asynchronously from grid point to grid point in steps. A busy robot will likely have to move off the grid to perform the given task. When the task is completed or cancelled, the robot starts a discovery phase immediately if it is located at a grid point; otherwise, it returns to the previous grid point before starting the discovery phase.

In addition to restricting the robots movement on a virtual grid, G-R3S2 further restricts robot movement by employing a Least Recently Visited (LRV) policy [2]. Simply speaking, a free robot is required to move to a least recently visited adjacent grid point rather than to a randomly selected one. Random choice is made only in case of a tie or in the event that there are no active sensors within the robots' communication radius. The LRV-based grid point selection obviously helps remove randomness in network exploration and improves the algorithm performance further. We shall elaborate on it below.

Each robot locally maintains an increasing sequence number (SeqNo), which will be used together with its ID to define a distinct visit to a grid point. It increases this number whenever its residence grid point changes. An off-grid robot keeps its last visited grid point as its residence grid point so that its returning visit (due to task completion or cancellation) is not counted. During a discovery phase, a robot broadcasts a beacon message carrying its ID, SeqNo, and grid location, and the message is replied by active sensors as in R3S2.

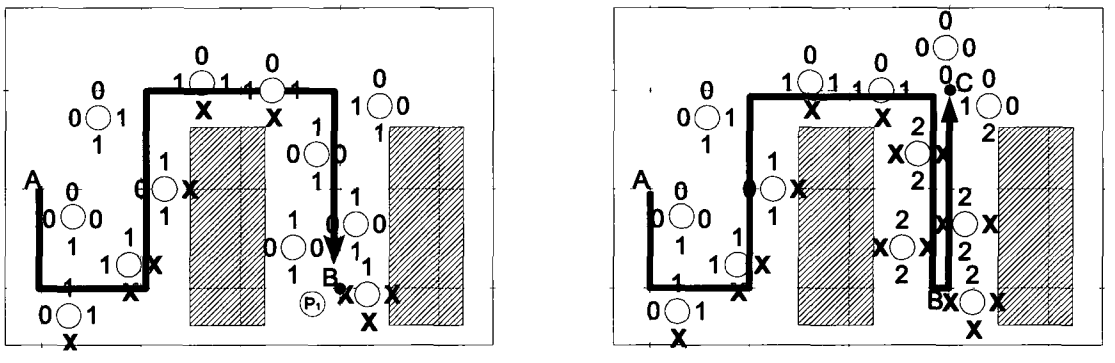
For each of its neighbouring grid points, an active sensor maintains a visit count and the last robot visit defined by the (robot ID, SeqNo) pair. It increases the visit count when a distinct robot visit to the point is observed. That is, when it receives a beacon message containing a (robot ID, SeqNo) pair different than the recorded last visit. When replying to the beacon message of a robot, an active sensor includes the adjacent grid point with smallest visit count in the report message. As such, after a discovery phase, a robot is able to identify the least recently visited neighbouring grid point and then move to it if free. The LRV policy implemented for G-R3S2 for the robots, as well as sensors is detailed in Alg. 4 and Alg. 5, respectively.

Algorithm 4. G-R3S2 LRV policy, robot movement and task selection

```
1:  if hole ≠ null and passiveSensors ≠ null then
2:      pick up passiveSensor and repair hole such that moving distance is minimized
3:      return to previous grid point
4:  else if hole ≠ null and passiveSensors = null then
5:      if carrying sensors then
6:              repair hole
7:              return to previous grid point
8:  else if hole = null and passiveSensors ≠ null then
9:      pick up with probability 1 − p
10:     return to previous grid point
11: else if sensor within communication range then
12:     move in direction d, recommended by sensor i
13: else
14:     move in random direction
15: end if
```

Algorithm 5. G-R3S2 LRV policy, sensor

```
1:  if redundant then
2:      delegate proxy
3:      "sleep"
4:  else
5:      transmit passive sensors and sensing holes
6:      transmit min(Weight(d))
7:      Weight(d) := Weight(d) + 1
8:  end if
```



(a) LRV Path to Redundant Sensor (b) LRV Path to Sensing Hole

Figure 17. Robot Traversal Using Least Recently Visited Policy

The robot navigation using the LRV policy is illustrated in Fig. 17. In the figures above, the circles that are completely filled represent active sensors; with the numbers around them representing the weight for a given direction and an X if it has been

determined that the given direction is obstructed by an ROI boundary or a physical obstacle. In Fig. 17(a), the robot starts at position A and is eventually lead to position B, following the recommended directions given by sensors along the path. At B the robot will move off of the grid and pick up the passive sensor, P_1. After picking up P_1 the robot will move back to its previous grid point and continue its movement. In Fig. 17(b) the robot moves from the grid point at B to the grid point at C, at which point it moves off of the grid and places P_1 in a reported sensing hole.

4.5 Cluster-Based R3S2

In R3S2 and G-R3S2, robots move randomly – with some added constraints in G-R3S2 – throughout the ROI communicating with nearby active sensors in order to find passive sensors and local sensing holes. To get more complete information about the location of passive sensors, as well as the position and size of sensing holes, we will further extend R3S2 with a cluster-based approach, C-R3S2. For this extension we will utilize *connected dominating sets* (CDS) to create the clusters, with sensors in the CDS acting as cluster heads and remaining sensors as cluster members. Recall from Sec. 4.1.1 the technique used to build the CDS.

In this case every sensor in the network will either be in the CDS or have a neighbour which is in the CDS. The decision by a sensor, S, of which cluster to become a member of is made in the following way. Given a set of cluster heads, C, which are neighbours of S, S will become a member of the cluster with head C_i if $ID(C_i) > max(ID(C_j))$ for all neighbouring cluster heads $j \neq i$.

Cluster heads receive information from their members indicating the position of sensing holes and passive sensors. Once they have received information from all of their members, cluster heads will merge local sensing hole information if the local holes are part of the same larger hole. Cluster heads then broadcast this information to all of their members, giving sensors knowledge of their entire cluster.

When local coverage situation changes, i.e., when a reported local hole (center) has been covered or when a neighboring active sensor fails, sensors notify the cluster head, which then requests cluster members (including the newly placed sensor) to re-send

their current list of local sensing holes. The cluster head will re-run the virtual force algorithm and broadcast the results to its cluster members.

The robots' movement policy remains unchanged, with the exception that encountered sensors will now exchange information about the position of sensing holes and passive sensors within their cluster. This will give the robots a more accurate view of the size and location of sensing holes, allowing the robots to repair the holes more efficiently and effectively. Additionally, the robots will follow the same deployment policy. Therefore there will be no additional computation cost for the robots under the cluster-based extension.

From above, we see the core of this cluster-based extension is local sensing hole merger. In the following we shall discuss how to accomplish this in detail.

4.5.1 Merging Local Sensing Holes

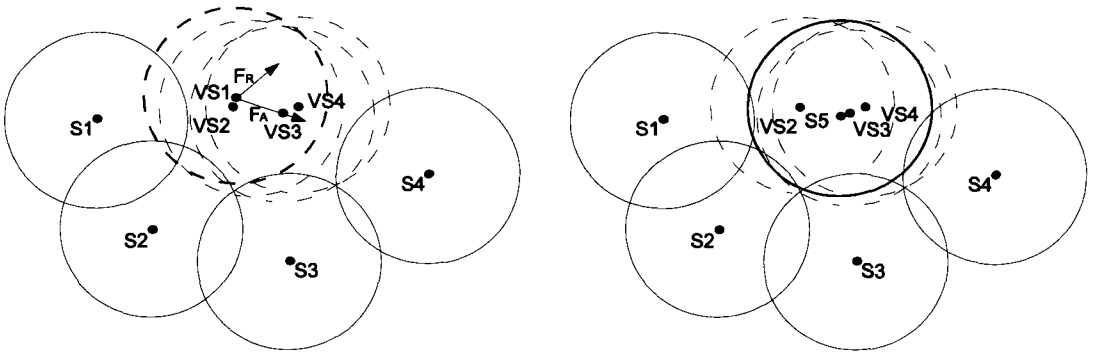
Recall from Sec. 4.1.2 the basic definition of the virtual forces algorithm from [44, 45]. We will use the virtual forces algorithm to merge the local sensing hole information for a cluster and re-compute the location where a replacement sensor should be placed. In this case the sensors deployed in the ROI are static, while the locations of local sensing holes (i.e., their centers) are treated as virtual mobile sensors. The mobile sensors will receive force contributions from obstacles, ROI boundaries, static sensors and the areas of preferential coverage (if any exist), with the force calculations described previously. As previously stated, the center of a sensing hole is the position symmetric to the location of a sensor, S, by the chord of the uncovered arc. When calculating the forces acting on a virtual mobile sensor (local sensing hole), the cluster head selects the distance from S to the center of the hole as the nodal separation, d_{th}. By merging local sensing holes, cluster members are able to recommend hole locations to robots such that the robots are able to perform hole healing more efficiently; that is, with fewer replacement sensors.

One method for re-computing the sensing hole locations is to perform a one-time re-computation. That is, the cluster heads consider all of the local sensing holes – virtual mobile sensors – at one time. The sensing holes would then receive force contributions from other holes, as well as deployed sensors and obstacles. There would not be any

areas of preferential coverage in this method, however. This method only requires the cluster heads to run the virtual forces algorithm once (in iterations); however, it may lead to inefficient sensor placement. Because the reported holes are based on only local information, there may be more reported holes than actual holes within the cluster as the number of virtual mobile sensors is not decreasing. Therefore, some holes should be removed from consideration.

Another method for re-computing the sensing hole locations is an incremental approach. That is, the cluster heads will consider one reported sensing hole at a time. In this case, the currently unconsidered holes can be treated as preferred coverage areas, generating attractive forces. For each reported sensing hole the cluster heads will ensure that the hole still exists prior to running the virtual forces algorithm (i.e. it has not since been covered by a re-located sensor from a previous execution of the algorithm) by checking if the centre of the hole is covered. If the hole no longer exists in the reported location, the hole is removed from consideration and thus no longer generates attractive forces. While it is possible that parts of a hole remain uncovered despite its centre being covered, the hole is removed from consideration nonetheless.

This incremental method will reduce the number of holes to re-compute iteratively and finally the number of hole locations to report to cluster members, thus reducing the message size and saving energy for communication. Additionally, this will reduce the number of required passive sensors to heal the holes within a cluster. For these reasons we choose to use this approach for re-computing sensing hole locations.



(a) Initial Hole Location (b) Final Hole Location

Figure 18. Sensing Hole Re-computation Using Virtual Forces

The virtual force algorithm for re-computing the location of sensing holes is illustrated in Fig. 5. In Fig. 5(a), the solid circles, labelled S_1, S_2, S_3, S_4, represent already-deployed sensors, while the dashed circles, labelled V_{S1}, V_{S2}, V_{S3}, V_{S4}, represent the virtual mobile sensors in the locations of their locally identified sensing holes. In this case the force contributions acting on V_{S1} are shown. S_1 does not generate any force as $d_{S1VS1} = d_{th}$. S_2 exerts a repulsive force F_R, while V_{S3} and V_{S4} act as areas of preferential coverage and thus exert a total attractive force F_A.

Figure 5(b) shows the sensor configuration after a sensor S_5 had been placed in the location previously calculated for V_{S1}. In this case, it can be seen that V_{S1} had been relocated such that all the reported local sensing holes turn out to be the same hole and could be filled with a single sensor. As previously stated, after V_{S1} had been re-located, V_{S2}, V_{S3} and V_{S4} would be removed from consideration. Without employing the virtual force algorithm to re-calculate sensing hole locations, it would have required more than one sensor to fill the hole, resulting in a less efficient use of passive sensors.

Chapter 5: Simulation Results

5.1 Sensor Placement

In this section we will evaluate BTD in comparison with SLD [6], [7] and LRV [2] through extensive simulation. This work has appeared in [12]. We use the following performance metrics.

- Coverage Ratio (CR): The average ratio of the number of occupied grid points to the total number of grid points at termination.
- Robot Moves (RV): The average number of movements made by each robot during the simulation.
- Robot Messages (RM): The average number of messages generated by each robot during the simulation.
- Sensor Messages (SM): The total number of messages transmitted by sensors during the simulation.

Recall that each robot moves the same distance, equal to the grid edge length, at each step. RV reflects both deployment latency and per robot energy consumption for movement. RM indicates per robot energy consumption for communication. SM implies the total energy consumption for communication among sensors. While CR should be maximized, the other three metrics are expected to be minimized for fast and efficient placement of sensors.

5.1.1 Simulation Setup

We implemented SLD, LRV and BTD with a custom network simulator. For a fair comparison, we slightly modified LRV in our implementation as follows. Each sensor periodically transmits "HELLO" message as it does in BTD, and uses the message to reply to any request from a robot. Then by listening to "HELO" message, robots receive recommended movement directions. We also extended LRV to multi-robot scenarios. Sensors do not distinguish robots; the visit of any robot to a grid point contributes to the visit count of that point. As LRV does not terminate itself, we consider it terminates as soon as each grid point has been visited by a robot.

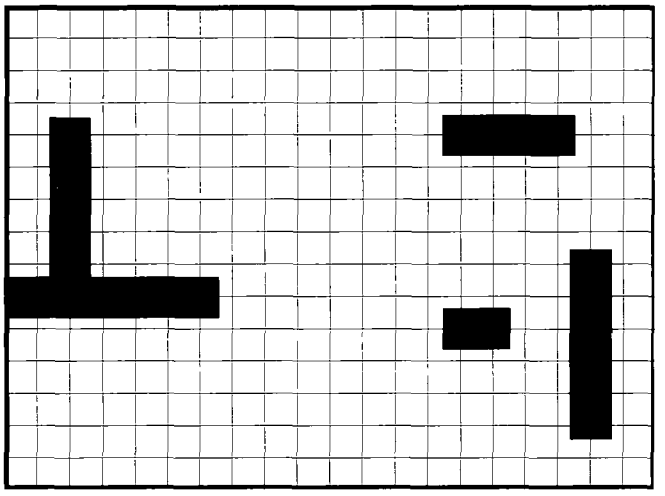


Figure 19. Simulation Environment for Sensor Placement

We simulated the execution of these algorithms in a rectangular ROI containing four obstacles in various shapes. Sensors have the same sensing radius $r_s = 15\sqrt{2}$ and the communication radius for sensors and robots is set to $r_c = 2r_s = 30\sqrt{2}$. Thus, a virtual grid of edge length $r_s\sqrt{2} = 30$ is established over the ROI, as shown in Fig. 19, where the black blocks represent obstacles. We place m robots initially at randomly selected grid points. The robots move asynchronously at a random speed between 0.1 and 1.1 and drop sensors at grid points. They have the same communication radius as sensors.

Because SLD does not guarantee coverage, we are interested only in the benefit of BTD over SLD in CR. We compared them in a failure-free environment with $m = 1$. This is because SLD does not support multiple robots or tolerate failure. For comparison between BTD and LRV, we conducted simulation both in a failure-free environment and a failure-prone environment where 7 sensing holes of average size h (i.e., occupying h grid points) occur randomly in time. The location and shape of the sensing holes are also determined randomly. This is done by randomly selecting the location of the first sensor within the hole to fail, and then a random adjacent sensor fails until h sensors have failed. If the current failure sensor has no adjacent sensors, sensors adjacent to the hole may fail until h sensors have failed or there are no more sensors adjacent to any of the failure locations. By selecting a random adjacent sensor at each step when creating the sensing holes, the shapes of the holes vary.

For the failure-prone and failure-free environments we investigated the impact of the number of robots on performance. Additionally, for the failure-prone environment we also investigated the impact of sensing hole size on performance. For these investigations we first varied m from 1 to 7 (while fixing h to 5, in the failure-prone environment). We then varied h from 1 to 9 (while fixing m to 3).

5.1.2 Coverage Ratio

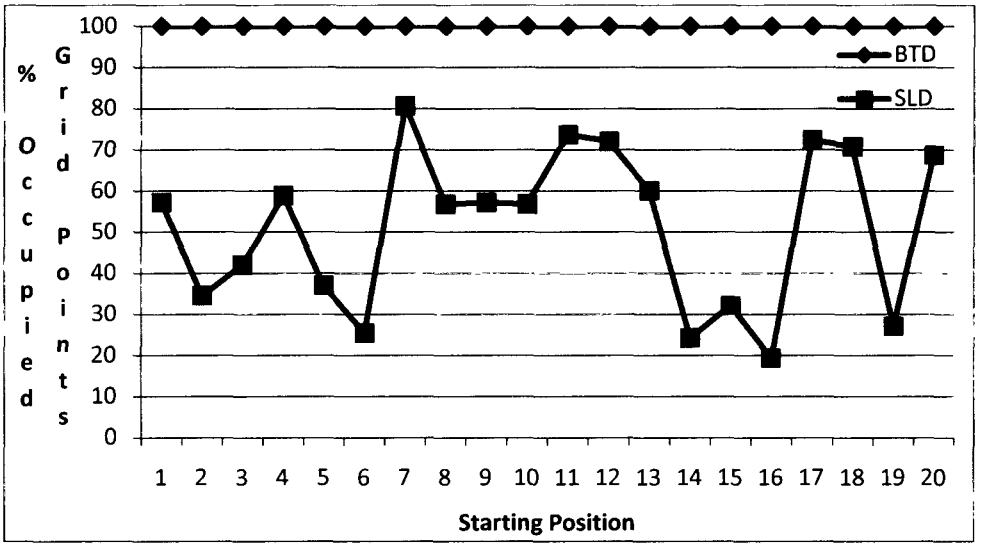


Figure 20. Coverage Ratio: BTD vs. SLD

We first study the performance of the test algorithms on CR. The simulation results are shown beginning with Fig. 20. According to Fig. 20, SLD generates 40% – 50% coverage over the ROI on average, with a variation from 20% to 80%. In fact, an arbitrarily low coverage ratio close to 0 may be possible for SLD in some cases, for example, in a large ROI where the robot quickly becomes stuck after it started to move. On the other hand, BTD guarantees full coverage, verifying Theorem 2.

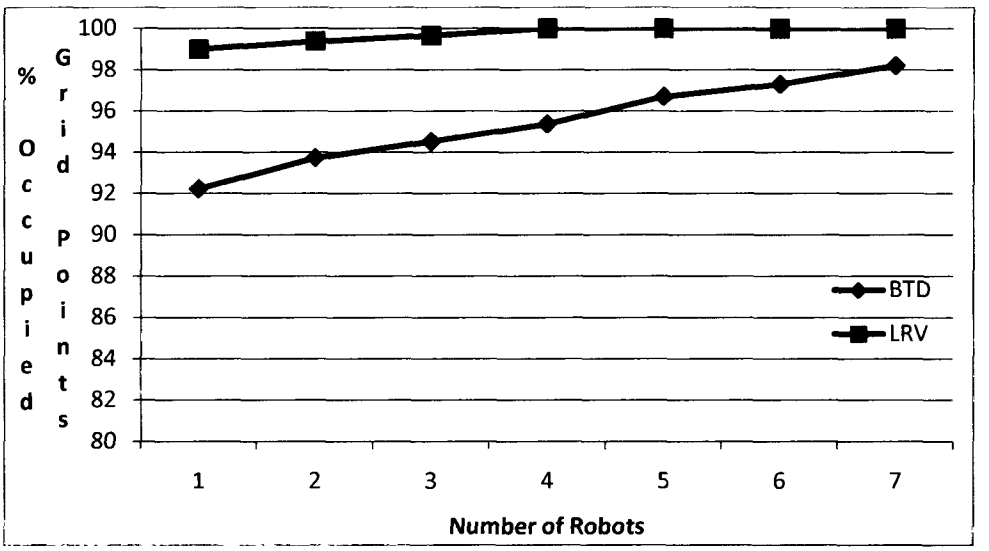


Figure 21. Impact of Number of Robots on Coverage Ratio

In BTD, sensing holes that do not affect robot back tracking are not known by the robots and thus left untreated. LRV does not produce full coverage either if sensing holes are not met by robots before the ROI is fully explored. Fig. 21 shows their CR in relation with *m*. When *m* increases, for both BTD and LRV there is an increasing trend in CR. This is expected because, as *m* increases the likelihood that a hole is encountered and fixed by a robot will increase. We notice that the curve of BTD is below that of LRV. It is because that randomized movement in LRV gives robots higher probability of discovering (fixing) sensing holes.

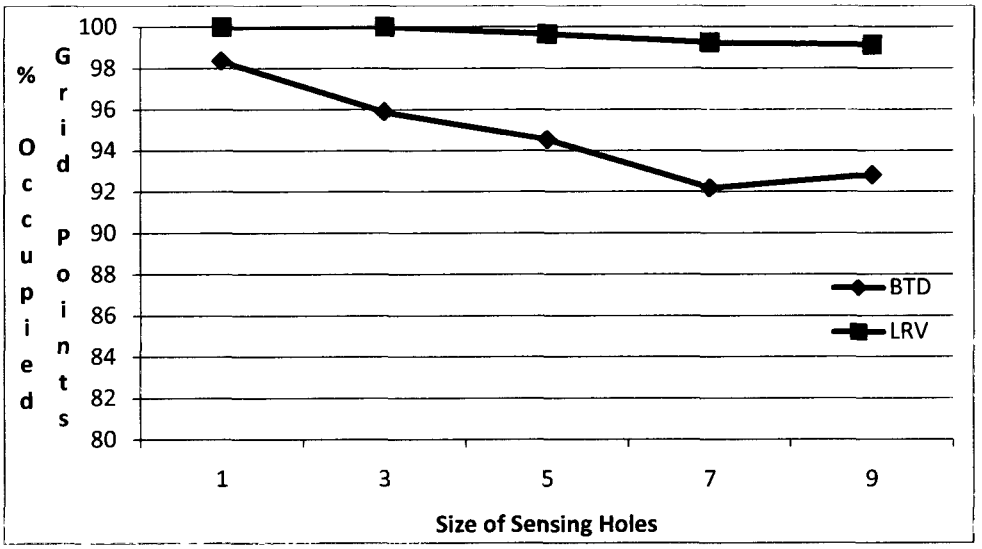


Figure 22. Impact of Size of Sensing Hole on Coverage Ratio

Fig. 22 shows the impact of h on CR. Due to BTD's nature (ignoring holes that do not impact ROI exploration), as h grows, CR steadily declines as more and more sensors are left unrepaired until h becomes relatively large compared to the environment. Once h is large enough, the robots begin to encounter more holes during ROI exploration, thus increasing CR. With LRV, CR decreases as h increases because LRV will not necessarily replace all of the failed sensors when a hole is encountered. Due to its random or movement it is likely that the holes will eventually be fixed, but as h grows, more sensors fail and the probability that the failed sensors will not be fixed before every grid point has been visited increases.

As shown in Figs. 21 and 22, LRV outperforms BTD in terms of CR in failure-prone environment; however, BTD still provides a satisfactory and very high CR (> 92%), and by allowing slightly lower CR it gets significant performance gains (> 80% savings) in other areas, as we will see from other simulation results below. In addition to its lack of the important terminatability property, this tells us LRV is not a good option for sensor placement or even coverage maintenance. A combination of our new algorithm BTD and an efficient separate coverage repair algorithm may be a practical solution.

We have seen that SLD does not provide any guarantee on the final coverage even in the ideal failure-free environment. It is indeed not a comparable algorithm to BTD and

therefore out of consideration in the remainder of the simulation study. Below we will elaborate our simulation results comparing BTD and LRV on movement and message cost. As we will see, BTD is much more efficient than LRV in RV and RM at no additional cost of SM in failure-free environment and at reasonable cost of SM in failure-prone environment.

5.1.3 Impact of Number of Robots

We first study the impact of number of robots, m, on the algorithm performance. Our simulation results for this purpose are depicted beginning with Fig. 23.

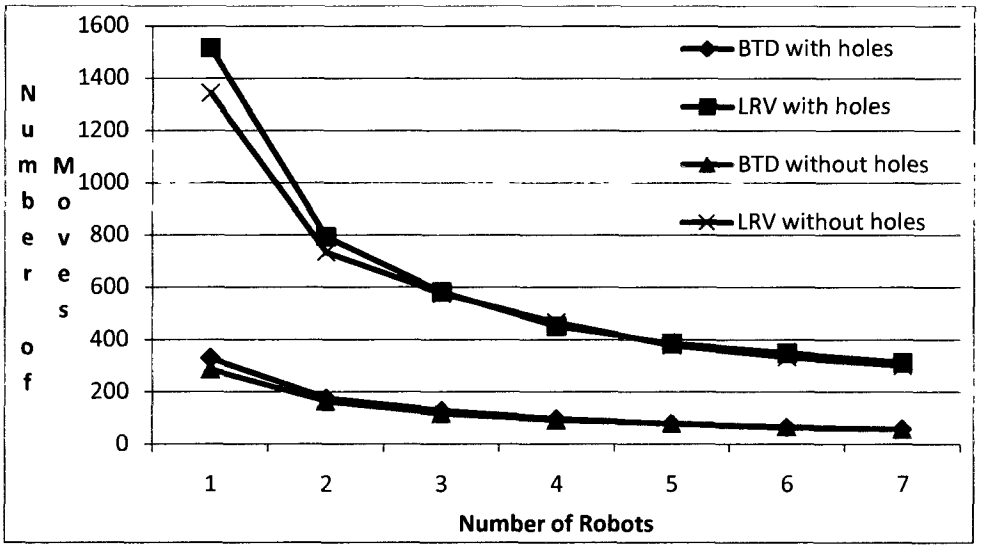


Figure 23. Impact of Number of Robots on Robot Moves

Fig. 23 illustrates the relation between RV and m. We notice a descending trend in both the LRV and BTD curves. This confirms the intuition that the more robots, the faster the algorithm terminates. We also observe that BTD far outperforms LRV, over 3 times more efficient. The degree to which BTD outperforms LRV can be seen more clearly by comparing the results for BTD when $m = 1$ to LRV when $m = 7$. In the case with no sensing holes, BTD terminates in fewer RV, despite utilizing only a single robot compared to the 7 used by LRV; in the case where sensing holes may occur, LRV terminates just in slightly fewer RV. The reason for such a significant difference is the number of unnecessary moves induced by randomness as part of LRV, whereas the

robots implementing BTD move in a more intelligent way by constantly attempting to move to the nearest empty location (with preference given to its own path).

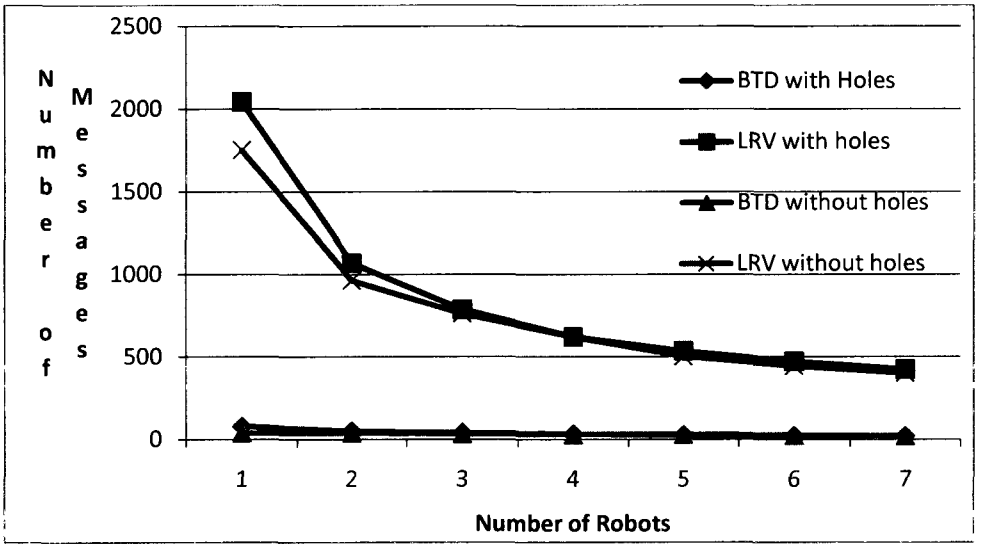


Figure 24. Impact of Number of Robots on Robot Messages

Fig. 24 illustrates how *m* affects RM. As with Fig. 23, we once again notice a descending trend. The reason for this trend in the case of LRV is that each robot, before moving to a chosen direction, sends a notification message to the sensor at its current location so that the latter can update the visit count of that direction. Therefore, as the number of total movements decrease, so too do the number of messages sent from robots. In the case of BTD, we notice a curve that is much less steep. This is because with the BTD algorithm robots only need to send messages when back-tracking, which represents a fairly small percentage of the robot movements. And, as the number of robots increases, each robot back-tracks less frequently on average because there is a greater chance that the unexplored areas along a robot path have been explored by other robots.

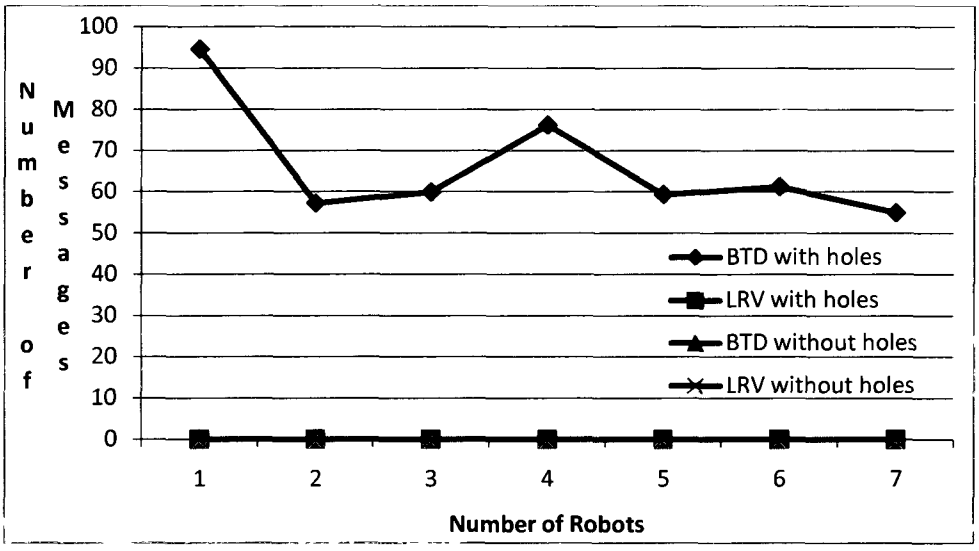


Figure 25. Impact of Number of Robots on Sensor Messages

Fig. 25 illustrates how SM varies with m. In LRV and in BTD with no sensing holes, sensors do not transmit any message in addition to "HELLO" messages (which are not counted). Thus, we observe the corresponding curves overlapping with the X axis. In the case of BTD with holes, robots send search message along the boundaries of encountered sensing holes for back-tracking recovery. We have known that the more robots the less frequently each robot performs back-tracking. If back-tracking is indeed needed, it may possibly occur early because there is a greater chance that a robot can get boxed in by sensors deployed by other robots, in addition to obstacles, ROI boundaries and the sensors it had placed itself. All of these factors lead to the decrease of the total number of encounters of robots and sensing holes. As a result, SM decreases as m increases, as shown in Fig. 25.

5.1.4 Impact of Size of Sensing Holes

Intuitively, the smaller the average sensing hole size h, the smaller the chance that a robot encounters a sensing hole during back-tracking, and therefore the less message cost induced. We can see from Figs. 23 – 25 that the difference in performance with and without sensing holes is negligible for both algorithms LRV and BTD with small size holes (h = 5). Below we will study the impact of sensing hole size. The simulation results are given beginning with Fig. 26.

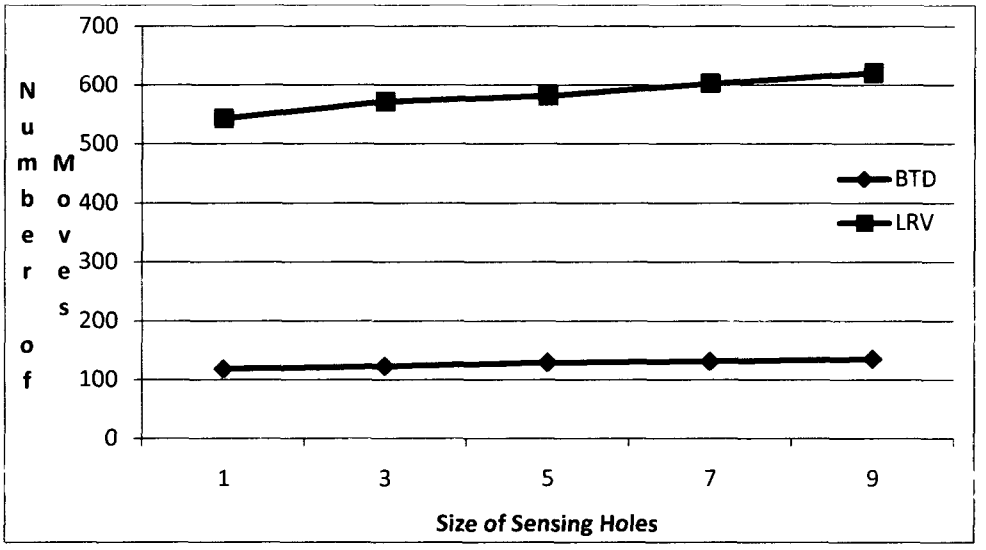


Figure 26. Impact of Size of Sensing Hole on Robot Moves

Fig. 26 shows RV versus *h*. For LRV we notice an ascending trend. This is because as the size of sensing holes increases there are more sensors that need to be replaced by the robots employing the LRV algorithm. Once a robot enters a sensing hole it will not move outside until all the replacement sensors are visited the same number of times as the sensors outside the hole. However, with BTD we notice very minimal change until the size of the sensing holes becomes greater than 5. This is because the larger the size of the sensing hole, the greater the probability is that the hole will be encountered by a robot during its exploration of the environment. As a result when a hole is encountered the robots must replace the failed sensors, and then resume the interrupted back-tracking. There is a slight increase in the moves required to terminate for BTD.

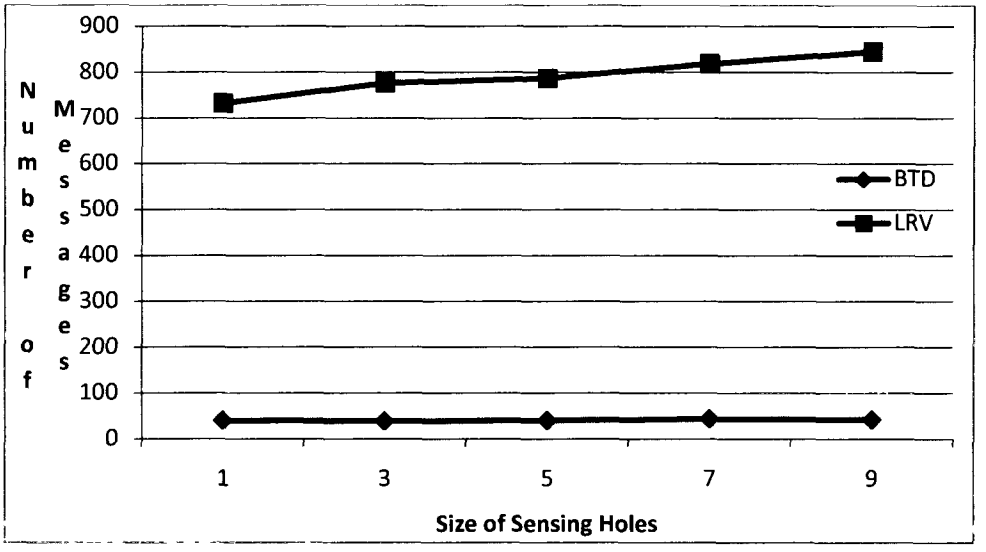


Figure 27. Impact of Size of Sensing Hole on Robot Messages

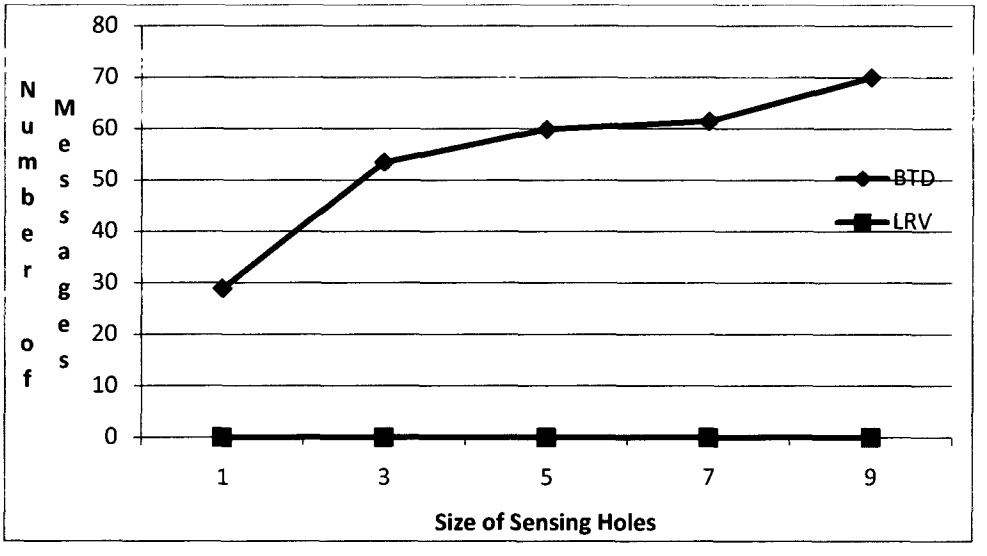


Figure 28. Impact of Size of Sensing Hole on Sensor Messages

Figs. 27 and 28 illustrate RM versus h and SM versus h, respectively. For LRV we once again notice an ascending RM and zero-value SM. As above, with larger sensing holes LRV requires more moves by each robot. As a result RM increases since LRV requires that prior to each movement step a robot must send a notification message to the sensor at its current location. SM remains 0 since no message is transmitted no matter how large the sensing holes are. For BTD we notice an almost flat trend in Fig. 27; however, as the size of the sensing holes increase there is a slight increase in RM. The

increase can be attributed to the robots encountering slightly more holes during their backtracking. The increase in RM and the growth of sensing hole size h together lead to big increase in SM, as shown by the ascending BTD curve in Fig. 28.

5.2 Sensor Relocation

In this section we evaluate R3S2, G-R3S2, C-R3S2 and C-G-R3S2 through extensive simulation. This work has appeared in [13]. We use the following performance metrics.

- Robot Moves (RM): The average number of movements made by each robot during the simulation.
- Total Moving Distance (MD): The average total distance traveled by each robot during the simulation.
- Bit Cost (BC): The average number of bits transmitted by each sensor replying to robot beacon messages.
- Repair Rate (PR): 1 – the ratio of uncovered area after repair over the initial uncovered area.

Both RM and MD are indicators to per robot energy consumption and coverage repair latency. Recall that there is a discovery phase for each robot between two successive movement steps. BC reflects per sensor energy consumption for communication (due to the tested algorithms). While these metrics should be minimized, PR should be maximized.

5.2.1 Simulation Setup

We implemented R3S2, G-R3S2, C-R3S2 and C-G-R3S2 with a custom network simulator, implemented in Java. The sensors and robots are initially placed randomly throughout a rectangular ROI containing six obstacles of various shapes, sizes and configurations. We set $r_s = 12\sqrt{2}$, $r_c = 2r_s = 24\sqrt{2}$ and $R_c = r_c = 24\sqrt{2}$. In G-R3S2 and C-G-R3S2, we chose to use a virtual square grid of edge length equal to $r_s\sqrt{2} = 24$. Fig. 29 shows the test ROI with the virtual grid drawn. The black blocks represent obstacles. As previously discussed, for C-R3S2 and C-G-R3S2, cluster heads select d_{th} to

be the distance from the sensor that reported the sensing hole to the center of the reported hole. By dynamically selecting the d_{th} value in this way, it ensures that a replacement sensor would cover the reported uncovered arc, while minimizing the amount of overlap between sensing radii – as discussed in Sec. 4.2 – in the event of a net zero force contribution from neighbouring obstacles and sensors.

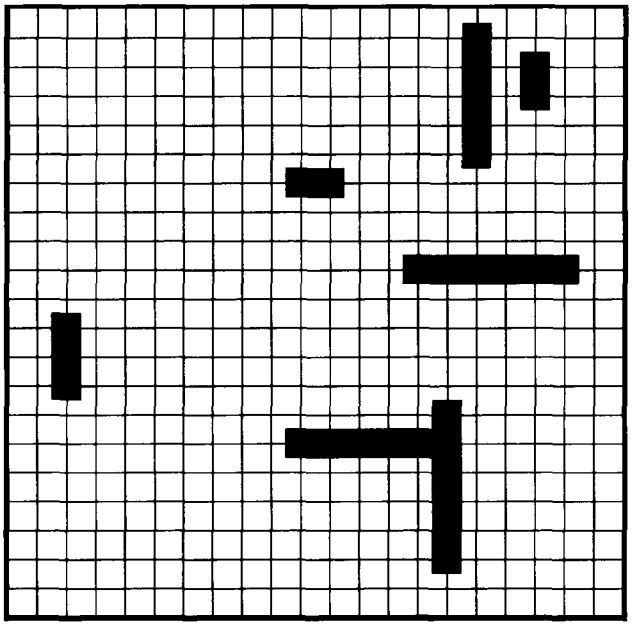


Figure 29. Simulation Environment for Sensor Relocation

Passive sensors are identified by the activity scheduling algorithm [15]; sensing holes are determined locally, following the method outlined in Sec. 4.2. Robots move asynchronously step by step. For each step, they move at a random speed between 0.1 and 1.1 and a distance of 24 (equal to the grid edge length). The length of the discovery phase between two movement steps is 100 simulated time units. The simulations terminate either once the robots have relocated all redundant sensors and none of the robots have any additional redundant sensors on hand or there are no more holes to fill.

To ease analysis, in simulation, we did not consider emerging holes caused by run-time sensor failure, although their treatment is automatically provided by the same algorithms. Thus, the locations and number of the passive sensors and sensing holes was determined randomly based on the initial sensor layout. We first varied the number of

robots, m, from 1 to 7, while fixing the number of sensors in the environment, n, to 400. We then varied n from 300 to 500, in increments of 50 sensors, while fixing the m to 3.

We will see in the following analysis that the grid-based extensions outperform those which do not have a virtual grid for the robot to move along. Additionally, we will see that the cluster-based extensions help to reduce the number of movements and message cost versus the non-cluster approach.

5.2.2 Impact of Number of Robots

We first study the impact of the number of robots, *m*, on the algorithm performance. Our simulation results for this purpose are shown beginning with Fig. 30.

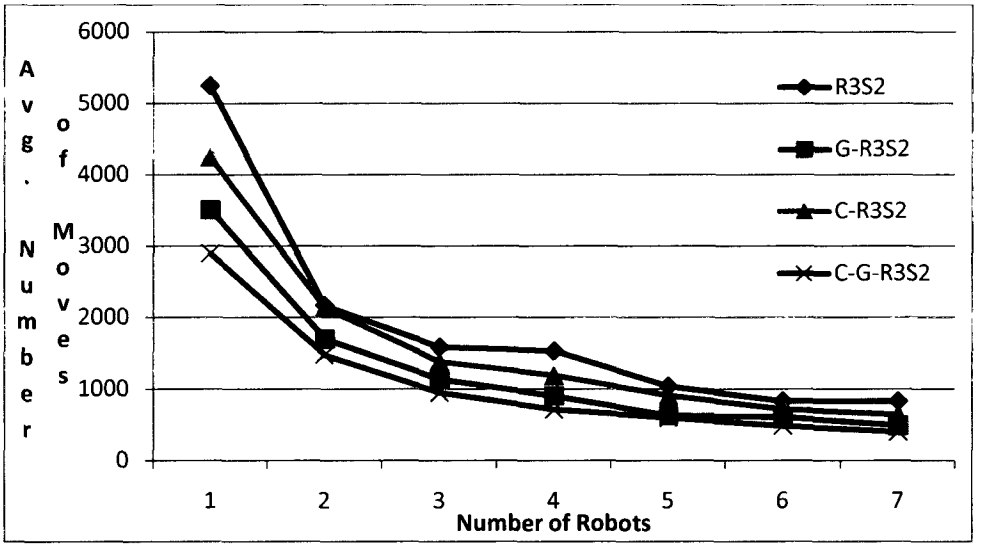


Figure 30. Impact of *m* on Robot Moves

Fig. 30 illustrates the relation between RM and *m*. We notice a decreasing trend in all curves. This confirms the intuition that the expected coverage repair delay time decreases as there are more robots serving the network. We also notice that as the number of robots increase the difference in performance is minimized. The reason for the large difference in performance with fewer robots is that the grid-based algorithms are more intelligent, as the robots move to directions visited less frequently and thus areas more likely containing passive sensors and sensing holes. However, as the number of robots increases, R3S2 and C-R3S2 increase their probability of finding passive sensors

and holes despite the random movement strategy. Also with few robots we notice the cluster-based variants outperform their non-cluster counterparts. This is due to sensors having more knowledge in the cluster extensions about passive sensors and sensing holes, rather than just neighbour information.

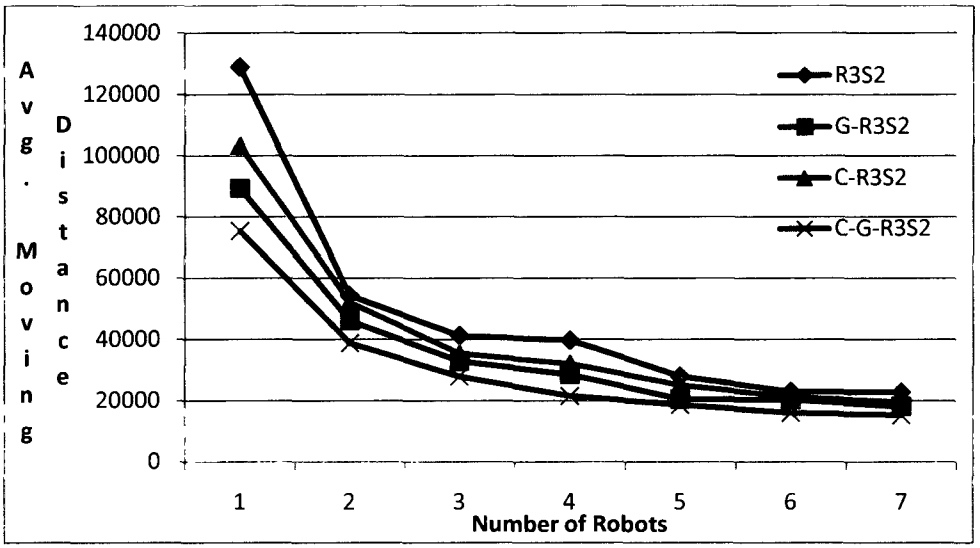


Figure 31. Impact of *m* on Total Moving Distance

Fig. 31 illustrates the relation between MD and *m*. As with RM, we notice a decreasing trend in all curves. Again we notice a difference in performance when there are five or fewer robots in the environment. As the number of robots increases, the difference in performance becomes less than the difference seen in Fig. 30. This is because G-R3S2 and C-G-R3S2 require extra moving distance when the robots move from a grid point to either pick up or drop a passive sensor (at an off-grid location) and then again to return to its previous grid point. As a result of this extra distance traveled the difference between the performance of the grid-based variants and the non-grid based variants with more than five robots is negligible in our simulations.

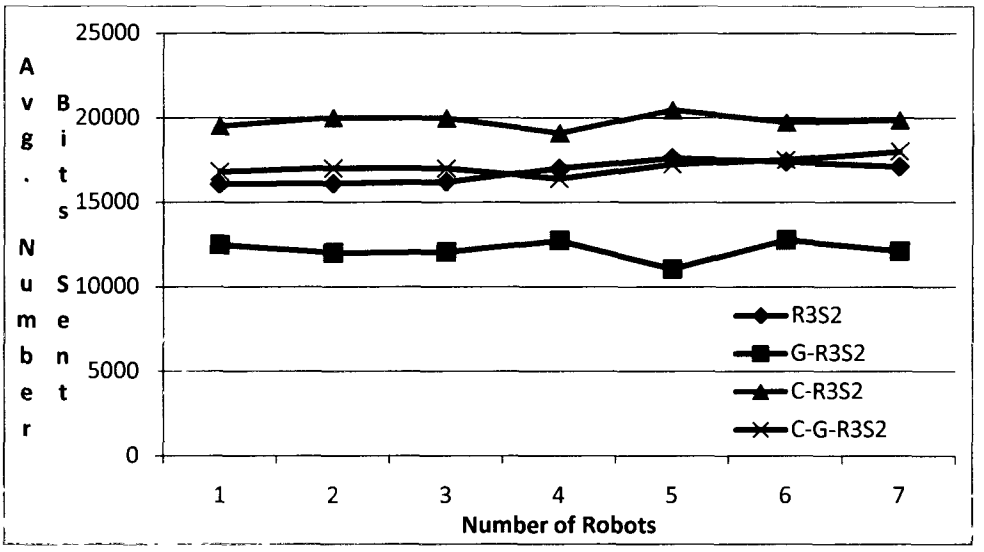


Figure 32. Impact of *m* on Bit Cost

Fig. 32 illustrates the relation between the BC and *m*. We notice a near flat trend for all simulations. This flat trend is expected. BC depends on the number of sensing holes and passive sensors in the environment and the number of times they are suggested to robots (as reply to a discovery phase). When n is fixed, the average number of holes and passive sensors is roughly fixed under random sensor distribution. The same sensing hole or passive sensor may be suggested more often when there are more robots. But the probability is very small or negligible as it is subject to the chance that multiple robots visit the same hole or sensor with random movement in a large area plus the change that the hole or sensor is not handled during previous robot visits. Additionally, in the cluster-based variants, BC is affected by the messages sent by cluster members and cluster heads when re-computing the sensing hole locations and notifying members of the locations of passive sensors. The difference in performance between the grid-based variants and non grid-based variants is due to the increased number of moves in the non-grid based variants, increasing the number of discovery phases and thus the BC for the replies. The reason for the higher BC in the cluster-based variants is due to the extra communication between cluster heads and cluster members that is necessary for all cluster members to be aware of the new hole locations and the location of all passive sensors within the cluster.

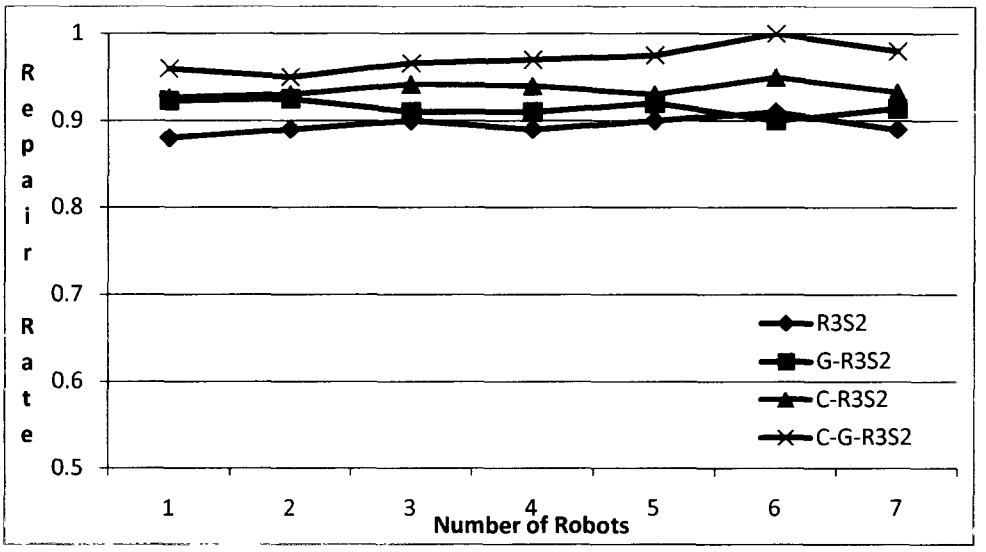


Figure 33. Impact of *m* on Repair Rate

Fig. 33 illustrates the relation between the PR and *m*. We again notice a flat trend for R3S2, as well as all of its variants. This trend is to be expected when n is fixed because of the random distribution of sensors and the localized nature of the algorithm. The number of passive sensors when n is fixed will be approximately the same for the simulations and due to the robot's random movement and access to only local information or information local to a given cluster, the passive sensors may not be placed optimally to heal the holes that exist in the network. Therefore we have relatively flat trends that are generally less than perfect, are still repairing at least 90% of the reported holes. However, the difference in PR when comparing the cluster based variant to its non-cluster based alternative (i.e. C-R3S2 vs. R3S2 and C-G-R3S2 vs. G-R3S2) we notice a significant improvement in PR. This is due to the fact that, while the robots still do not have global knowledge of the environment, they are able to fill the encountered sensing holes more intelligently due to the cluster heads re-calculating the sensing hole locations and informing the other sensors in the cluster of the new locations.

5.2.3 Impact of Number of Sensors

We now examine the impact of the number of sensors, n, on the algorithm performance. Our simulation results for this purpose are shown beginning with Fig. 34.

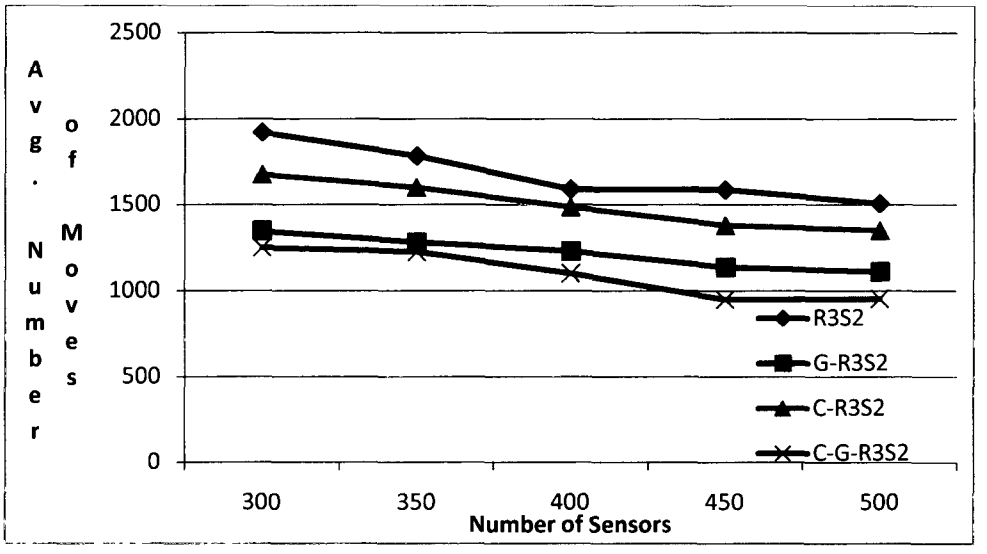


Figure 34. Impact of *n* on Robot Moves

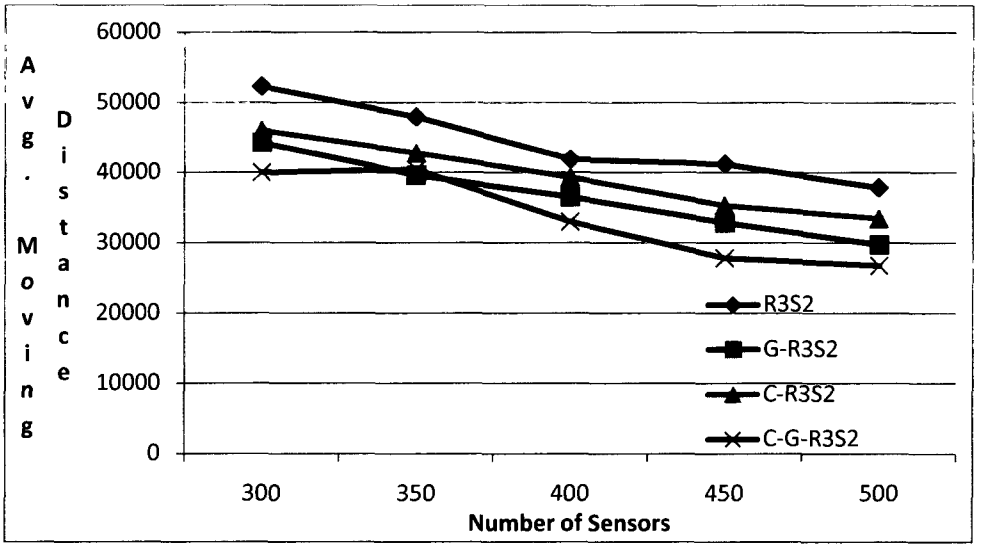


Figure 35. Impact of *n* on Total Moving Distance

In Figs. 34 and 35 we observe the relation between RM and MD with n, respectively. In both figures we notice a monotonically decreasing trend for both algorithms. Once again the grid-based variants are seen to have the best overall performance, while the cluster-based variants outperform their non-cluster counter. The difference in performance between the algorithms remains near constant regardless of the number of sensors deployed. The reason for the decreasing trend is that – despite the stochastic nature of the sensor deployment – as the number of sensors in an environment

increase, it is likely that the number of coverage holes will decrease and the number of passive sensors will increase. Therefore, with fewer holes to heal and more passive sensors to relocate the number of movements by each robot and the distance traveled by each robot decrease.

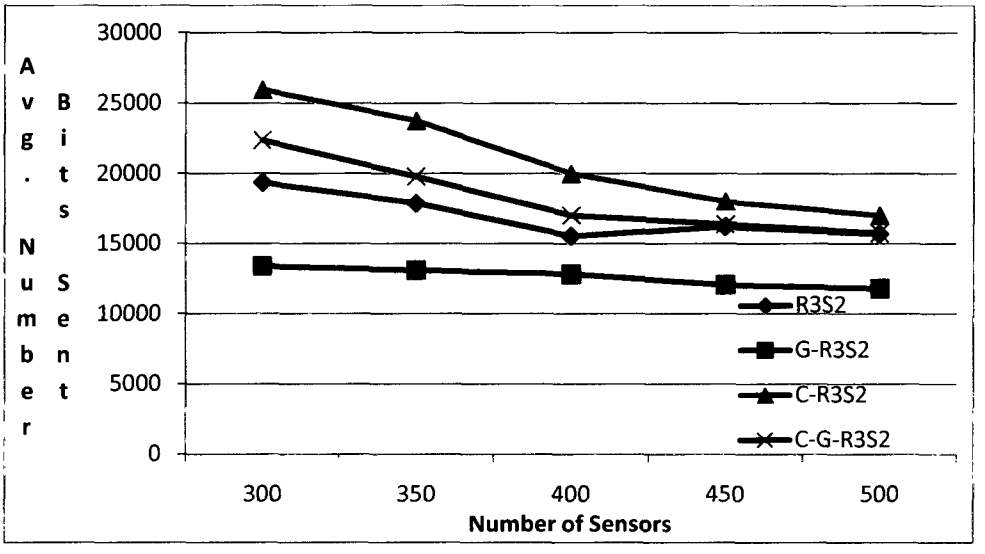


Figure 36. Impact of *n* on Bit Cost

Fig. 36 illustrates the relation between BC and *n*. For G-R3S2, C-R3S2 and C-G-R3S2, we again notice a monotonically decreasing trend. For R3S2 we notice an overall decreasing trend with a larger than expected drop in bit cost for the 400 sensor scenario. Again, we notice higher BC for the cluster-based variants. This is due to the extra communication that is not required in the non-cluster variants. As previously discussed the number of bits transmitted will decrease as the number of robot movements decreases.

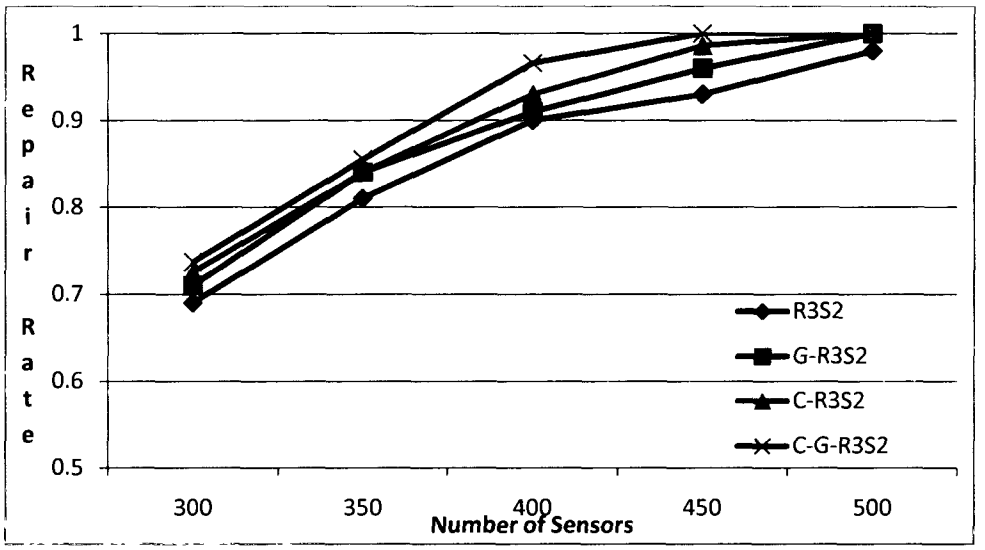


Figure 37. Impact of *n* on Repair Rate

Fig. 37 illustrates the relation between PR and *n*. In this case we notice a monotonically increasing trend for R3S2, G-R3S2, C-R3S2 and C-G-R3S2. This trend is expected, as we would expect that as the number of sensors increases the number and size of sensing holes decreases and the number of passive sensors increases. Therefore we have more sensors to fill the declining number of sensing holes. We notice that at 400 sensors, all of the tested algorithms achieve a repair rate of at least 90% and as the number of sensors continue to increase the repair rates trend to 100%. Similar to the case in Fig. 33 we again notice a difference in the PR when comparing the cluster based variants against the non-cluster based variants.

Chapter 6: Conclusions and Future Work

This thesis addressed both the problem of carrier-based sensor deployment and the problem of carrier-based sensor relocation with an emphasis on finding localized solutions in both cases. We first presented a novel localized solution for the sensor placement problem, namely Back-Tracking Deployment (BTD). We demonstrated BTD with both single-robot case and multi-robot case. We proved the correctness of BTD in a failure-free environment, and presented a fault tolerant approach in a failure-prone environment. Our simulation results indicate that BTD far outperforms the only competing algorithm LRV [2] in various metrics in addition to outperforming an incomplete algorithm in terms of coverage area in a failure-free environment. Further we addressed specific issues which may have prevented BTD from terminating successfully or which may have had a negative effect on performance. Namely, we addressed the problem of how to recover from cyclic sensing holes and provided a load balancing solution when multiple robots are deploying sensors.

We then presented a novel localized solution to the problem of carrier-based sensor relocation, namely Randomized Robot-assisted Relocation of Static Sensors (R3S2). We then extended R3S2 introducing a grid-based variant (G-R3S2), which employs a *Least Recently Visited* policy to limit randomness in the robots' movements, which was shown to improve efficiency of the algorithm. We further extended R3S2 to include a cluster-based variant (C-R3S2 and C-G-R3S2), which provided the robots with more accurate information of the ROI without resorting to a centralized algorithm. This further improved the efficiency and effectiveness of the proposed algorithm.

As previously stated, to the best of our knowledge R3S2 and all of its variants are the first localized algorithms for relocation of static sensors for coverage repair, as existing coverage repair algorithms require robots to initially carry sufficient spare sensors [33] (no sensor relocation), require each sensor to have locomotion (mobile sensor) [29, 30, 41] or require centralized control [11]. We demonstrated R3S2, G-R3S2, C-R3S2 and C-G-R3S2 using a single robot case, as well as a multiple robot case. Through extensive simulations, we showed that all of the aforementioned algorithms provide reasonable results across all measured metrics.

In future work we look to extend our proposed algorithms in the following ways:

- We will explore ways in which the robots move in a more intelligent way, rather than random movement for R3S2 and its variants (see the comparison between BTD and LRV [2] in Sec. 5.1.3 and Sec. 5.1.4).
- We will compare the performance tradeoffs (if any) from running BTD using different virtual grid types (i.e. square grid vs. triangle tessellation vs. hexagonal grid).
- We will explore further fault tolerant techniques for BTD in order to recover data from failed sensors.
- We will continue to explore further load balancing techniques for both algorithms.

References

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

[2] M. A. Batalin and G. S. Sukhatme. "Coverage, Exploration and Deployment by a Mobile Robot and Communication Network". Telecommunication Systems, 26:181–196, 2004.

[3] P. Bose, P. Morin, I. Stojmenovic and J. Urrutia. "Routing With Guaranteed Delivery in Ad Hoc Wireless Networks". In Proc. of ACM Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), LNCS 4325, pp. 48–55, 1999.

[4] W. Burgard, M. Moors, D. Fox, R. Simmons and S. Thrun. "Collaboartive Multi-Robot Exploration". In Proc. of IEEE International Conference on Robotics and Automation (ICRA), pp. 476–481, 2000.

[5] J. Carle and D. Simplot-Ryl. "Energy Efficient Area Monitoring by Sensor Networks". IEEE Computer Magazine, 37(2): 40-46, 2004.

[6] C.Y. Chang, C.T. Chang, Y.C. Chen and H.R. Chang. "Obstacle-Resistant Deployment Algorithms for Wireless Sensor Networks". IEEE Tran. on Vehicular Technology, 58(6): 2925–2941, 2009.

[7] C.Y. Chang, J.P. Sheu, Y.C. Chen and S.W. Chang. "An Obstacle-Free and Power-Efficient Deployment Algorithm for Wireless Sensor Networks". IEEE Tran. on System, Man and Cybernetics – Part A: Systems and Humans, 39(4): 795–806, 2009.

[8] F. Dai and J. Wu. "Distributed Dominant Pruning in Ad Hoc Networks". In Proc. of IEEE International Conference on Communications (ICC), pp. 353–357, 2003.

[9] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". Numerische Mathematik, 1: 269–271, 1959.

[10] M. Dorigo and T. Stützle. "Ant Colony Optimization". MIT Press, Cambridge, MA, 2004.

[11] R. Falcon, X. Li, A. Nayak and I. Stojmenovic. "The One-Commodity Traveling Salesman Problem with Selective Pickup and Delivery: an Ant Colony Approach". In Proc. of IEEE Congress on Evolutionary Computing (CEC), 2010.

[12] G. Fletcher, X. Li, A. Nayak and I. Stojmenovic. "Back-Tracking Based Sensor Deployment by a Robot Team". In Proc. of IEEE Sensor Mesh and Ad Hoc Communications and Networking (SECON), pp. 1–9, 2010.

[13] G. Fletcher, X. Li, A. Nayak and I. Stojmenovic. "Randomized Robot-Assisted Relocation of Sensors for Coverage Repair in Wireless Sensor Networks". In proc. of IEEE Vehicular Technology Conference (VTC-Fall), 2010. To appear.

[14] K. R. Gabriel and R. R. Sokal. "A New Statistical Approach to Geographic Variation Analysis". Systematic Zoology, 18: 259–278, 1969.

[15] A. Gallais, J. Carle, D. Simplot-Ryl and I. Stojmenovic. "Localized Sensor Area Coverage With Low Communication Overhead". IEEE Tran. on Mobile Computing, 7(5): 661–672, 2008.

[16] F. Garcia, I. Stojmenovic and J. Zhang. "Addressing and Routing in Hexagonal Networks With Applications in Location Update and Connection Rerouting in Mobile Phone Networks". IEEE Tran. on Parallel and Distributed Systems, 13(9): 963–971, 2002.

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

[18] M.L. Garey and D.S. Johnson. "Computers and Intractability: A Guide to the Theory of NP-Completeness". W.H. Freeman, San Francisco, 1979.

[19] H. Hernández-Pérez and J. J. Salazar-González. "The One-Commodity Pickup-and-Delivery Travelling Salesman Problem". In Proc. of Combinatorial Optimization – Eureka, You Shrink!, LNCS 2570, pp. 89–104, 2003.

[20] J. Hightower and G. Borriello. "Location Systems for Ubiquitous Computing ". IEEE Computer, 34(8): 57–66, 2001.

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

[22] A. Howard, M. J. Mataric and G. S. Sukhatme. "Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem". In Proc. of International Symposium on Distributed Autonomous Robotics Systems (DARS), pp. 299–308, 2002.

[23] F. Ingelrest, N. Mitton and D. Simplot-Ryl. "A Turnover Based Adaptive HELLO Protocol for Mobile Ad Hoc and Sensor Networks". In Proc. of IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 9–14, 2007.

[24] X. Li, H. Frey, N. Santoro and I. Stojmenovic. "Localized Sensor Self-Deployment with Coverage Guarantee". ACM SIGMOBILE Mobile Computing and Communications Review, 12(2): 50–52, 2008.

[25] X. Li, H. Frey, N. Santoro and I. Stojmenovic. "Focused Coverage by Mobile Sensor Networks". In Proc. of IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS), pp. 466–475, 2009.

[26] X. Li, H. Frey, N. Santoro and I. Stojmenovic. "Localized Sensor Self-Deployment for Guaranteed Coverage Radius Maximization". In Proc. of IEEE International Conference on Communications (ICC), pp. 1–5, 2009.

[27] X. Li, N. Mitton, I. Ryl and D. Simplot. "Localized Sensor Self-Deployment with Coverage Guarantee in Complex Environment". In Proc. of AdHoc-Now, LNCS 5793, pp. 138–151, 2009.

[28] X. Li, A. Nayak, D. Simplot-Ryl and I. Stojmenovic. "Sensor Placement in Sensor and Actuator Networks". Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication, Wiley, 2010.

[29] X. Li and N. Santoro. "ZONER: A ZONE-Based Sensor Relocation Protocol for Mobile Sensor Networks". In Proc. of IEEE Conference on Local Computer Networks (LCN), pp. 923–930, 2006.

[30] X. Li, N. Santoro and I. Stojmenovic. "Mesh-based Sensor Relocation for Coverage Maintenance in Mobile Sensor Networks". In Proc. of Ubiquitous Intelligence and Computing (UIC), LNCS 4611, pp. 696–708, 2007.

[31] X. Li, N. Santoro and I. Stojmenovic. "Localized Distance-Sensitive Service Discovery in Wireless Sensor and Actor Networks". IEEE Tran. On Computers, 58(9): 1275–1288, 2009.

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

[33] Y. Mei, C. Xian, S. Das, Y. Charlie Hu and Y.H. Lu. "Sensor Replacement Using Mobile Robots". Computer Communications, 30(13): 2615–2626, 2007.

[34] K. Menger. "Zur allgemeinen Kurventheorie". Fund. Math., 10: 96–115, 1927.

[35] K. Menger. "Das Botenproblem". Ergebmisse Eines Mathematischen Kolloquiums 2, pp. 11–12, 1932.

[36] A. Nayak and I. Stojmenovic. Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication, Wiley, 2010.

[37] S. Poduri, S. Pattern, B. Krishnamachari and G.S. Sukhatme. "Using Local Geometry for Tunable Topology Control in Sensor Networks". IEEE Tran. on Mobile Computing, 8(2): 218–230, 2009.

[38] L.C. Shiu. "The Robot Deployment Scheme for Wireless Sensor Networks in the Concave Region". In Proc. of IEEE International Conference on Networking, Sensing and Control (ICNSC), pp. 581–586, 2009.

[39] I. Stojmenovic. "Localized network layer protocols in wireless sensor networks based on optimizing cost over progress ratio". IEEE Network 20(1): 21–27, 2006.

[40] G. Voronoi. "Nouvelles applications des paramètres continus à la théorie des formes quadratiques". Journal für die Reine und Angewandte Mathematik, 133: 97–178, 1907.

[41] G. Wang, G. Cao, T. La Porta and W. Zhang. "Sensor Relocation in Mobile Sensor Networks". In Proc. of IEEE International Conference on Computer Communications (INFOCOM), pp. 2302–2312, 2005.

[42] B. Yamauchi. "A Frontier-Based Approach for Autonomous Exploration". In Proc. of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), pp. 146–151, 1997.

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

[44] Y. Zou and K. Chakrabarty. "Sensor Deployment and Target Localization Based on Virtual Forces". In Proc. of IEEE International Conference on Computer Communications (INFOCOM), pp. 1293–1303, 2003.

[45] Y. Zou and K. Chakrabarty. "Sensor Deployment and Target Localization in Distributed Sensor Networks". ACM Tran. on Embedded Computing Systems, 3(1): 61–91, 2004.