

# Back-Tracking based Sensor Deployment by a Robot Team

Greg Fletcher, Xu Li\*, Amiya Nayak, and Ivan Stojmenovic  
 SITE, University of Ottawa, Canada  
 gflet062@uottawa.ca, {xuli, anayak, ivan}@site.uottawa.ca

**Abstract**—Existing solutions to carrier-based sensor placement by a single robot, in a bounded unknown region of interest (ROI), do not guarantee full area coverage and/or do not terminate. We propose a novel localized algorithm, named *Back-Tracking Deployment* (BTD). To construct a full coverage solution over the ROI, mobile robots (carriers) carry static sensors as payload and drop them at visited empty vertices of a virtual square, triangular or hexagonal grid. A Single robot will move forward along the virtual grid in open directions with respect to a pre-defined order of preference until a dead end is reached. Then it back-tracks to the nearest sensor adjacent to an empty vertex on its backward path (it is an ‘entrance’ to an unexplored/uncovered area). The robot resumes regular forward moving and sensor dropping from there. To save movement steps, the back tracking is performed along a locally identified shortcut. We extend the algorithm to support multiple robots that move independently and asynchronously. Once a robot reaches a dead end, it will back track, giving preference to its own path. Otherwise it will take over the back-track path of another robot, by consulting with neighboring sensors. We prove that BTD terminates in finite time and produces full coverage when no sensor failures occur. We also describe an approach to handle sensor faults. 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.

## I. INTRODUCTION

Wireless sensor networks are large-scale static wireless multi-hop networks [17] where nodes have limited resources such as energy, bandwidth, storage, and processing power. They are dedicated to provide low-level surveillance and data gathering services normally in a region of interest (ROI) for various high-level applications such as traffic analysis, habitat study, and intrusion detection. To this end, they must maximally or fully cover the ROI without internal sensing holes. Sometimes, additional requirement on node degree [15], node density [9] or coverage focus [12], for example, may apply. It can not however, be expected that sensors are placed in a desired way as they are often randomly dropped due to operational factors such as human inaccessibility to the ROI and tight deployment budget.

Wireless sensor and robot (also called actuator or actor) networks (WSRN) [14] are an integration of wireless sensor networks and multi-robot systems. They consist of networked sensor and robot nodes that communicate via wireless links to perform distributed sensing and actuation tasks. Robots are relatively resource-rich, usually mobile and are involved in making decisions and performing appropriate actions on

themselves (e.g., controlled movement), sensors (e.g., repairing failed sensors) and/or the environment (e.g., stopping a fire). In WSRN, the impact on coverage from stochastic node dropping, coupled with controlled mobility and actuation of robots, gives rise to the problem of *movement-assisted sensor placement* for coverage formation.

There are different ways to place sensors by exploiting mobility in WSRN, depending on the concrete network model. Robots may carry static sensors as payload and move around in the ROI. While traveling, they deploy sensors at proper positions (e.g., vertices of certain geographic graph). We call this method *carrier-based*. In a special case of WSRN, mobile (robotic) sensors may play the robot role. In this case, a sensor *self-deployment* approach may be engaged. By this approach, sensors autonomously and intelligently change their geographic location, adjusting the overall distribution to a desired one. We have investigated sensor self-deployment in our earlier work [12]. In this paper, we concentrate our attention on carrier-based sensor placement.

### A. Problem statement

A number of mobile robots, pre-loaded with static sensors, are scattered randomly in an unknown bounded 2-dimensional ROI (e.g., terrain and boundary information is not known by robots a priori). They are aware of their own position by attached GPS devices or other positioning techniques, and are able to detect obstacles and boundaries of the ROI by attached scanning laser range-finder, for example. Sensors have equal sensing radii  $r_s$ . Sensors and robots are equipped with the same communication radii  $r_c$ . They may communicate directly as long as they are within each other’s communication range. Suppose that each robot carries sufficient 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.

We assume  $r_c \geq 2r_s$  (see [7] for a discussion). Sensor placement to achieve full coverage and connectivity was studied in [1]. Square grid sensor placement produces full coverage when nodal separation is at most  $\sqrt{2}r_s$ . Sensor placement at vertices of regular triangles (triangular grid) produces full coverage with larger nodal separation  $\sqrt{3}r_s$ , thus with fewer sensors on the ground. Triangular grid can be obtained from a square grid by translating every even row by half the nodal distance. This operation does not impact our algorithm in any essential way, and we keep square grid in the algorithm description for simplicity, especially in illustrations. Hexagonal placement [8] offers similar efficiency (number of sensors needed to cover same area) as triangular one (it can be obtained from

\*Corresponding author

triangular one by eliminating a third of nodes, and by moving nodes to internodal distances  $r_s$ ). Hexagonal placement can be implemented using triangular pattern, where every third node in each row is not used (and distances are adjusted).

We require sensors to be placed at the vertices of a square grid, defined by a given orientation, edge length  $\sqrt{2}r_s$  and with  $(0, 0)$  being vertex. We ignore boundary effects as they have no impact on the algorithm design in any essential way except that a robot may need to drop an extra sensor (if necessary) when its movement is obstructed by ROI boundary/obstacles. We further require each deployed sensor to periodically transmit a ‘hello’ message carrying its position and other information necessary for the deployment algorithm. ‘Hello’ message is a basic networking tool for neighborhood discovery [11] 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 neighbors and any neighborhood change (e.g., node failure and node status change). By listening to ‘hello’ messages, robots are able to detect previously deployed sensors.

The algorithm should enable robots to avoid physical obstacles during the course of 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, communication should be reduced to a minimum as well. Thus we seek for localized solutions, which rely only on some available local knowledge, without resorting to global network information.

There are two existing solutions to the stated problem. Batalin and Sukhatme [2] proposed the Least Recently Visited (LRV) algorithm, where already deployed sensors give recommendations to robots for the direction to continue sensor placement. The algorithm produces full sensing coverage in a long run; but it uses excessive movements to explore the ROI, and does not even offer termination. Chang et al. [5], [6] presented a Snake-Like Deployment (SLD) algorithm. However, despite the claim made, this algorithm can not guarantee full area coverage. This motivates us to develop an algorithm that does guarantee coverage, terminate, and remain efficient in robot movement. Further, the algorithm is extended from single robot to robot team deployment, which was not considered in the literature.

## B. Our contributions

We present a novel localized solution, named *Back-Tracking Deployment* (BTD), to the above stated problem. Each robot aligns with a common virtual square grid and moves forward from grid point to grid point, independently and asynchronously to unobstructed directions in a pre-defined order of preference. During the forward moving, it drops sensors at grid points and associates them with an increasing sequence number. Each deployed sensor colors itself *white* if it is adjacent to an empty grid point, and *black* otherwise. It includes the color information in ‘hello’ message. The location of a white sensor propagates along the forward path of its dropping robot via ‘hello’ message to the next white sensor.

This enables all sensors to dynamically maintain a back pointer pointing to the nearest white sensor (if any exists) along the backward path of their dropping robot.

A robot reaches a *dead end* if all four directions are obstructed. In this case, it back-tracks to the nearest white sensor (entrance to an unexplored/uncovered area) on its own path according to the back pointer stored on the sensor at its current location. When a single robot deploys sensors, it can maintain back pointers in its own memory. However, when a team of robots are dropping sensors, the back pointer of a robot may point to a sensor that has been visited in the meanwhile by another robot, and a new pointer is needed. Therefore the sensors need to store the back pointers. Further, to avoid robot moving back to several such already explored areas, sensors may self-organize to update these pointers dynamically.

Back-tracking is performed by a shortcut approach: the robot moves to the neighboring sensor with the same back pointer and smallest sequence number. Then it resumes forward moving from there. When a robot reaches a dead end, it may also back track along another robot’s path, if there are no back pointers along its own path, using information stored at sensors. We prove that the algorithm terminates in finite time and guarantees full coverage in failure-free environment. We present a fault-tolerant approach to deal with sensing holes and back pointer loss due to sensor failures. Each robot treats encountered sensing holes during back-tracking as a new bounded ROI and recursively runs BTD to fill them. After hole filling, it resumes the interrupted back-tracking or starts a new one from a sensor on the hole boundary.

Through extensive simulation, we evaluate the performance of BTD in comparison with SLD [5] and LRV [2]. SLD does not support multiple robots or tolerate sensor failures. Thus the comparison with it is for failure-free single-robot scenarios. In our examples, SLD generates 40% - 50% coverage over the ROI on average, with big variation from 20% to 80%. On the other hand, BTD guarantees full coverage. 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 consider it terminates once the ROI is fully explored. We show that, compared with LRV, BTD has > 80% savings in robot moves and robot messages at no cost in failure-free environment and at minimal cost of < 0.4 messages per sensor and < 8% sensing coverage in the presence of failures.

The rest of this paper is organized as follows. We first give a brief literature review in Sec. II. Then we propose BTD and prove its correctness in the scenario with no sensor failures in Sec. III and discuss how to deal with failures in Sec. IV. After that, in Sec. V, we present our simulation results. We conclude the paper and discuss future work in Sec. VI.

## II. RELATED WORK

The key problem associated with carrier-based sensor placement is that the carriers, i.e., robots, must fully explore the ROI, in order to construct a full coverage. This is related to the map exploration problem in the traditional mobile robot field. Map exploration has been long studied both for single robot

scenarios [18] and for multiple robot scenarios [4]. However, these algorithms are not applicable to WSRN mainly due to model differences. For example, in a mobile robot system, robots typically have vision, but no communication, and make protocol decision by observing other robots' movement; however, for WSRN, typically robots have communication ability, but no vision. Also, these algorithms usually require unrealistic perception range, for example, each robot is able to 'see' the entire environment which is potentially very large.

Movement-assisted sensor deployment was studied in the literature in a very limited context. Majority of the previous works addressed sensor self-deployment for mobile sensor networks. Only a few shed light on the carrier-based deployment problem. In this section, we will review these algorithms in brief. They have major drawbacks such as inefficiency in communication, unclear or lack of terminating conditions and/or full coverage guarantee, even in failure-free environment. A comprehensive survey can be found in our recent article [13].

Chang et al. [5], [6] presented a deterministic snake-like deployment approach (SLD). The robot moves step by step along a pre-computed graph leading to optimal coverage, each step to an adjacent empty vertex in the graph according predefined rules, and drops a sensor after each step. With differently defined vertex selection rules, the robot trajectory will exhibit different patterns such 'S' shape [5] and spiral shape [6]. SLD is very likely to be stuck at dead ends due to obstacles or early dropped sensors, as discussed in [13]. Dead-end recovery was however not discussed; consequently, the algorithm does not guarantee full coverage. It does not support multiple robots either. Shiu [16] induced the lack of coverage guarantee of this algorithm to concave regions and suggested to treat each concave region as a separate environment. The obstruction from obstacles and sensors is overlooked, and coverage guarantee is still not accomplished.

Batalin and Sukhatme [2] proposed the Least Recently Visited (LRV) algorithm, a solution in which the sensors store weights for each direction that a robot can travel in. The weights represent the number of times that a robot has visited each direction for a given sensor. The sensors then recommend the direction with the lowest weight for the robot to travel or the direction that is the least recently visited. We will show in section V that LRV requires many unnecessary movements to fully explore a given ROI. These extra movements lead to an extremely large number of messages sent from the robots. As with SLD, it is unclear under what conditions LRV terminates. Further, it was presented with a single robot. Multi-robot extension was mentioned, but for a different purpose.

Howard et al. [10] presented an incremental approach for sensor self-deployment. It relies on a central controller to gather information of previously deployed sensors and select best deployment point for the next sensor. Mobile sensors therefore deploy themselves one at a time to push the frontier of the network forward to explore the unknown ROI. The algorithm can easily be used for carrier-based sensor placement. In this case, the central controller simply informs robots to move to drop a sensor at a specified location. Because of the

centralized nature of this algorithm, it is very expensive in bandwidth and energy usage for communication.

### III. 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. IV, 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, 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.

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 an already 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. 1, 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 single-robot scenario and then examine the more complex situation where multiple robots are present.

#### A. 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 (by 'hello' message), sensors know who is the successor/predecessor along the robot path among neighbors. 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'

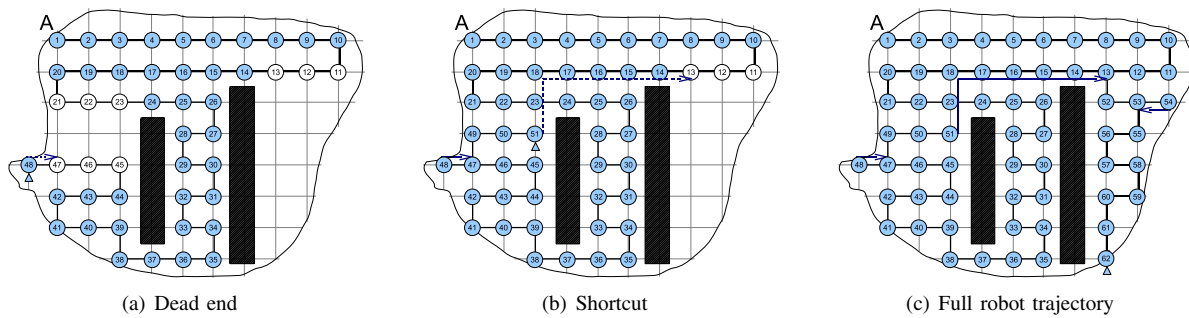


Fig. 1. BTD with single robot. Robot starts from position A. Its trajectory is shown by thick lines. Its current position is marked by a small triangle. Arrowed lines indicate back-tracking shortcut. Circles represent sensors. Numbers enclosed by circles are sequence number. Back pointers are omitted.

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.

In Fig. 1(a), white node 13 includes its color and back pointer (to node 12) in ‘hello’ message. After black node 14 receives the message, it sets up its back pointer to white 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 does the same. 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 Fig. 1(a) and 1(b), the robot’s back-tracking destinations are respectively node 47 and node 13. In Fig. 1(c), the robot stops moving 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 current 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 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. 1(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. Figure 1(c) shows the full robot trajectory.

### B. 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 can not 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 neighborhood and back track to the indicated white sensor. This is, in effect, back-track of another robot, which has been therefore ‘stolen’. 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.

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. 2, 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. The successor node (*a*, 27), after receiving the message, erases the back pointer from its memory and includes the erase information 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.

### C. Proof of correctness

**LEMMA 1:** Each grid point is visited at most 7 times overall, and at most 4 times by the same robot.

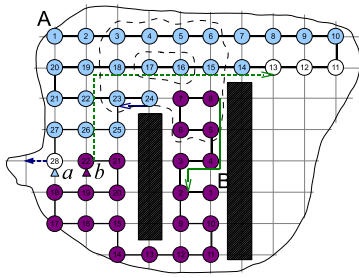


Fig. 2. BTD with multiple robots. Two robots  $a$  and  $b$  start respectively from position  $A$  and  $B$ .

*Proof:* Every time when a robot reaches a grid point, the robot visit count of 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 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. 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, 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 finite number of moves between grid points. This completes the proof. ■

**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 colored 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. ■

#### IV. 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 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.

A sensing hole is *acyclic* if the causal failure sensors themselves, when functioning properly, constitute a connected network without internal sensing hole, and *cyclic* otherwise. In Fig. 2, for example, if sensors inside the big dashed circle fail, an acyclic sensing hole will appear; if only sensors located between the two dashed circles fail, a cyclic sensing hole will emerge. We here assume acyclic sensing holes only and leave cyclic sensing hole situations for future work.

##### A. Finding recovery agent

To handle the sensing hole problem, we introduce a new color, *gray*. Sensors adjacent to a sensing hole (i.e., failed sensors) color themselves gray. The sensing hole is bounded by the gray sensors, the adjacent obstacles and the ROI boundaries, as illustrated in Fig. 3(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.

If all the failed neighboring sensors of current gray sensor have been replaced, the robot will consider the hole is filled and try to recover the back-tracking shortcut via current gray sensor, called *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. 3(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. Then 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

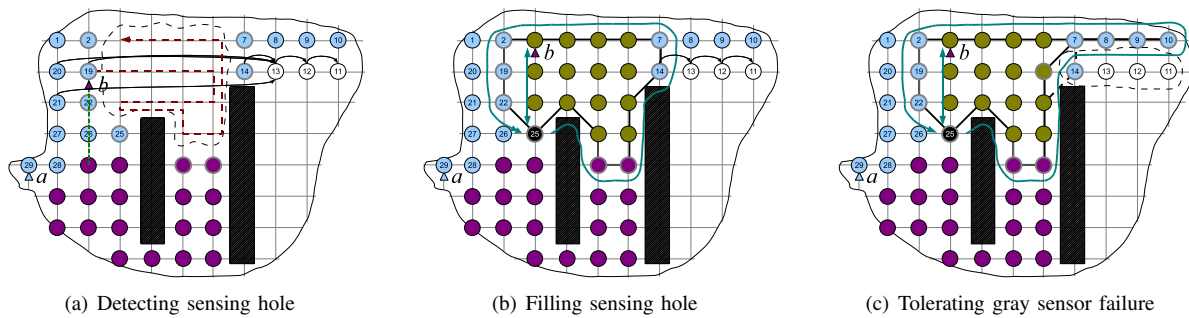


Fig. 3. Fault tolerance. Back pointers are drawn as thin arrowed line. Sequence numbers of sensors dropped by robot  $b$  are omitted. Dashed circle indicates sensing hole. Arrowed dashed line shows robot trajectory for sensing hole filling. Thick arrowed lines are message transmission path. Thick non-arrowed lines highlight the ring network around sensing hole. Black nodes are recovery agent.

Fig. 3(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 bounding obstacles and the ROI boundaries constitute a *ring network*, as illustrated in Fig. 3(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 starts a new back tracking. We will present the search process in the next sub-section.

### B. Searching 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 retrieval 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 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.

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 GRG principle. The objective is to tolerate gray node failure. For example, in Fig. 3(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 discovered back pointer. During the relocation to the destination, the robot may meet another sensing hole. For example, in Fig. 3(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.

## V. PERFORMANCE EVALUATION

In this section we will evaluate BTM in comparison with SLD [5] and LRV [2] through extensive simulation. We use the following performance metrics.

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

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

### A. Simulation setup

We implemented SLD, LRV and BTM 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 BTM, and uses the message to reply to any request from a robot. Then by listening to ‘hello’ 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.

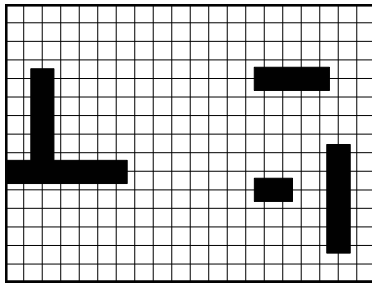


Fig. 4. Simulation environment

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 same communication radius  $r_c = 2r_s = 30\sqrt{2}$ . Thus a virtual grid of edge length  $r_s\sqrt{2} = 30$  is established over the ROI, as shown in Fig. 4, where the black blocks represent obstacles. We place  $m$  robots initially at randomly selected grid points. These 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 BTM over SLD in CR. We compared them in a failure-free environment with  $m = 1$ . It is because SLD does not support multiple robots or tolerate failure. For comparison between BTM and LRV, we conducted simulation both in a failure-free environment and a failure-prone environment where 7 sensing holes of size  $h$  (occupying  $h$  grid points) occur randomly in time, in location, and in shape.

A sensing hole is generated as follows. We first randomly select the location of the first sensor within the hole to fail, then a random adjacent eligible sensor is chosen to fail until  $h$  sensors have failed. If the current failure sensor has no adjacent eligible sensors, eligible sensors adjacent to the hole may fail until  $h$  sensors have failed or there are no more eligible sensors adjacent to any of the failure locations. A sensor is eligible for hole generation if its failure does not make the hole cyclic. By selecting a random adjacent eligible sensor at each step when creating the sensing holes, the shape 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). We ran the simulation 20 times for every scenario in both the failure-free and failure-prone environments in order to obtain the average results.

### B. Coverage ratio

We first study the performance of the test algorithms on CR. The simulation results are given in Fig. 5. According to Fig. 5(a) SLD generates 40% - 50% coverage over the ROI on average, with big variation from 20% to 80%. In fact, an arbitrarily low coverage ratio close to 0 may be possible for SLD in some cases, e.g., in a large ROI where the robot

quickly becomes stuck after it started to move. On the other hand, BTM guarantees full coverage, verifying Theorem 2.

In BTM, 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. Figure 5(b) shows their CR in relation with  $m$ . When  $m$  increases, for both BTM 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 BTM is below that of LRV. It is because that randomized movement in LRV gives robots higher probability of discovering (fixing) sensing holes.

Figure 5(c) shows the impact of  $h$  on CR. Due to BTM'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.

By Fig. 5(b) and 5(c), LRV outperforms BTM in terms of CR in failure-prone environment; however, BTM 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 BTM and an efficient separate coverage repair algorithm may be a practical solution.

### C. Movement and message cost

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 BTM and therefore out of our consideration in the rest of the simulation study. Below we will elaborate our simulation results about BTM and LRV on movement and message cost. As we will see, BTM 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.

1) *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 in Fig. 6.

Figure 6(a) illustrates the relation between RV and  $m$ . We notice a descending trend in both the LRV and BTM curves. This confirms the intuition that the more robots, the faster the algorithm terminates. We also observe that BTM far outperforms LRV, over 3 times more efficient. The degree to which BTM outperforms LRV can be seen more clearly by comparing the results for BTM when  $m = 1$  to LRV when

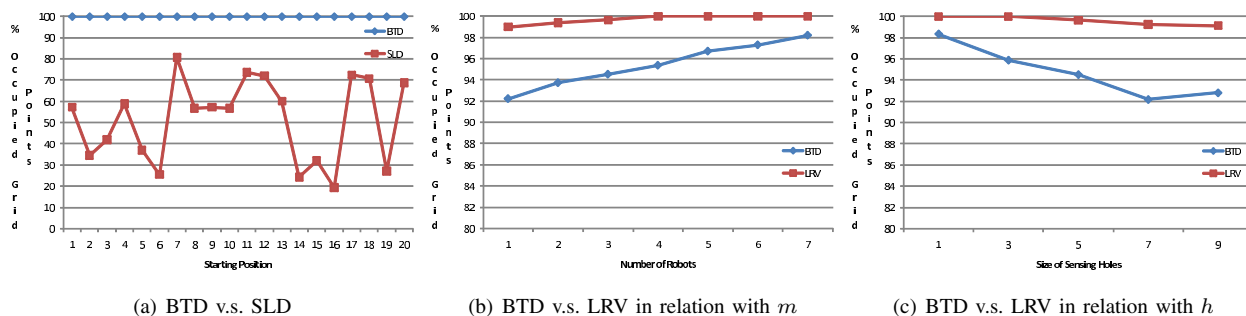


Fig. 5. Coverage ratio (CR)

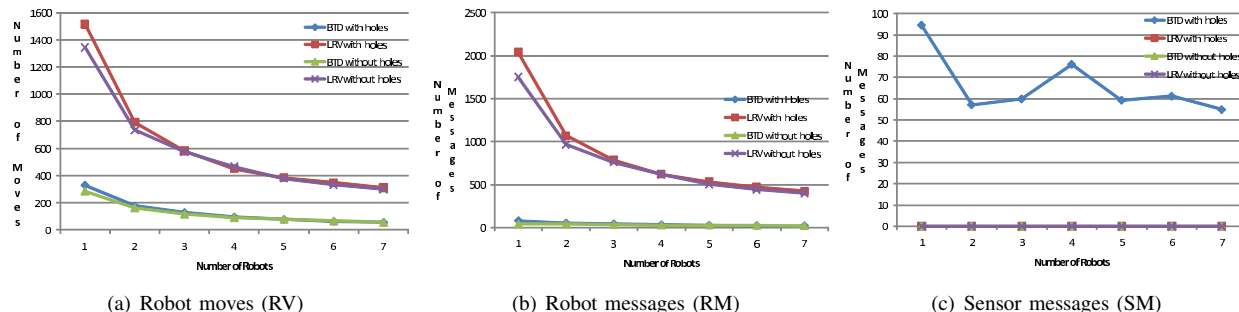


Fig. 6. Impact of  $m$  on movement and message cost

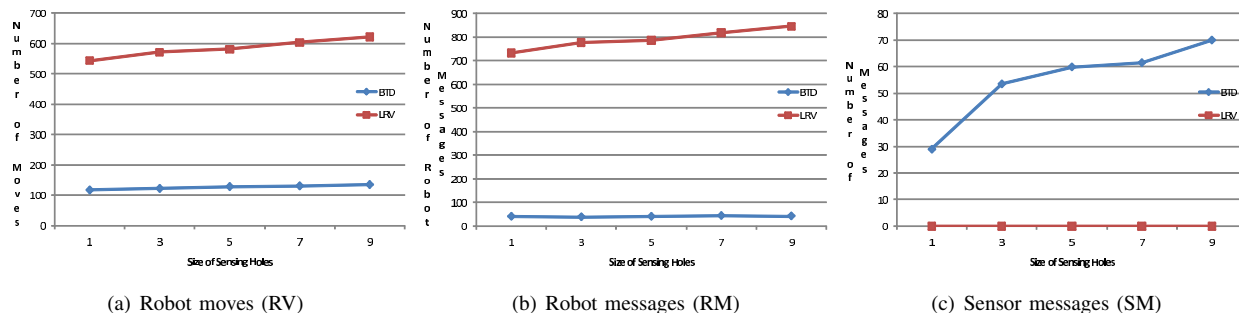


Fig. 7. Impact of  $h$  on movement and message cost

$m = 7$ . 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 6(b) illustrates how  $m$  affects RM. As with Fig. 6(a), 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 decreases, 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 6(c) 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’ message (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 these together 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 the figure.

2) *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 Fig. 6 that the difference in performance with and without sensing holes is negligible for both algorithms LRV and BTM with small size holes ( $h = 5$ ). Below we will study the impact of sensing hole size. The simulation results are given in Fig. 7.

Figure 7(a) 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 BTM 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, then resume the interrupted back-tracking. There is a slight increase in the moves required to terminate for BTM.

Figure 7(b) and 7(c) 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 BTM we notice an almost flat trend in Fig. 7(b); 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 back-tracking. The increase in RM and the growth of sensing hole size  $h$  together lead to big increase in SM, as shown by the ascending BTM curve in Fig. 7(c).

## VI. CONCLUSIONS

We presented a new localized algorithm, *Back-Tracking Deployment* (BTM), for exploring an unknown bounded ROI and deploying sensors. We demonstrated BTM with both single-robot case and multi-robot case. We proved the correctness of BTM in a failure-free environment, and presented a fault tolerant approach in a failure-prone environment. Our simulation results indicate that BTM far outperforms the only competing algorithms LRV [2] in various metrics. In the future we will improve BTM from the following considerations.

In the presence of sensor failures, BTM may lose its ability to terminate in some rare situations. For example, when a robot is placing sensors in a loop and sensors are failing from the tail of the deployment trace, the robot never meets a dead end and will not terminate unless it runs out of sensors. Additional, possibly not localized, technique is needed to ensure *terminatability*.

We have assumed acyclic sensing holes. In the case of *cyclic sensing holes*, only back pointers stored on the outer hole boundary should be followed for back tracking, and using a back pointer on the inner hole boundaries may lead to a

dead end and failure to fully explore the ROI. However, a robot is not able to differentiate the outer boundary and inner boundaries using only local knowledge. The presented fault-tolerance approach needs to be revisited.

When multiple robots are placing sensors, it is possible that some robots (e.g., robot  $a$  in Fig. 3) stop moving early during the deployment because of dead end and back pointer elimination by another robot, while others (e.g., robot  $b$  in Fig. 3) continue to explore almost the entire ROI. This *load balancing* problem needs to be addressed. Solutions would benefit sensor placement in many ways including balancing robot energy usage and reducing deployment latency.

## ACKNOWLEDGMENT

This work was partially supported by NSERC Strategic Grant STPSC356913-2007B.

## 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 DIALM, LNCS 4325, pp. 48-55, 1999.
- [4] W. Burgard, M. Moors, D. Fox, R. Simmons, S. Thrun. "Collaborative Multi-Robot Exploration". In Proc. of IEEE ICRA, pp. 476-481, 2000.
- [5] 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.
- [6] 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 Systems, Man, and Cybernetics-Part A: Systems and Humans, 39(4): 795-806, 2009.
- [7] 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.
- [8] 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.
- [9] M. Garetto and M. Gribaudo and C.-F. Chiasserini and E. Leonardi. "A Distributed Sensor Relocation Scheme for Environmental Control". In Proc. of IEEE MASS, pp. 1-10, 2007.
- [10] A. Howard, M.J. Mataric, and G.S. Sukhatme. "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks". Autonomous Robots, 13(2): 113126, 2002.
- [11] 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 MASCOTS, pp. 9-14, 2007.
- [12] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. "Focused Coverage by Mobile Sensor Networks". In Proc. of IEEE MASS, pp. 466-475, 2009.
- [13] 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.
- [14] A. Nayak and I. Stojmenovic. Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication, Wiley, 2010.
- [15] S. Poduri and S. Pattern and 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.
- [16] L.C. Shiu. "The Robot Deployment Scheme for Wireless Sensor Networks in the Concave Region". In Proc. of IEEE ICNCS, 2009.
- [17] J. Wu and I. Stojmenovic. "Ad Hoc Networks". IEEE Computer, pp. 29-31, 2004.
- [18] B. Yamauchi. "A Frontier-Based Approach for Autonomous Exploration". In Proc. of IEEE CIRA, pp. 146-151, 1997.