Online Multi-Robot Exploration with Communication Restrictions

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Elizabeth Anne Jensen

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Advisor: Dr. Maria Gini

August, 2018
Acknowledgements

This dissertation was a huge undertaking, and I am immensely grateful to my family and friends for their support over the years. Particularly to my parents, for reading many drafts and listening to the rough versions of my presentations, and for cheering me on when I wasn’t sure I would finish. To my friends, for listening when I ranted about robots not doing what I wanted, and helping distract me when I needed a break.

Thank you to my advisor, Maria Gini, for her guidance in all aspects of my graduate career. She gave me the freedom to explore what I wanted to do at my own pace, but also let me know when I got too far off track from the main goal (this thesis). She provided me with so many opportunities to grow and learn how to become the teacher and mentor that I want to be, through research. Being able to spend so much time learning how to teach students at all levels was invaluable to me. Her guidance on submitting papers and her tact in replying to reviews was always helpful.

Thank you also to my committee members, John Carlis, Stephen J. Guy, Wilma Koutstaal, and Nikolaos Papanikolopoulos, for your thoughtful comments and support even as I took longer than expected to complete this. I greatly appreciated your input.

I had many collaborators over the course of my work, and each of them helped
me learn and improve my research in some way. To my coauthors, Sara Lahr, Michael Franklin, Ernesto Nunes, London Lowmanstone, and Dr. Ken Sugawara, my sincere thanks.

Lastly, to the other graduate students who helped me along the way, Maitreyi Nanjanath, Brett Hemes, Duc Fehr, Will Groves, Mohamed Elidrisi, Mark Valovage, Julio Godoy, Marie Manner, Nick Johnson, James Parker, Steve Damer, Ian de Silva, Lian Duan, Daniel Kluver, Erik Steinmetz, and so many more. Your feedback on my research as it progressed, and our many conversations in and out of the lab were as fun as they were productive.
Dedication

To Mom, Dad, and Kat.
Abstract

In the aftermath of a disaster, such as an earthquake, it can be hours or even days before human rescue teams can safely enter damaged structures to search for and extract survivors. This critical time can be productively utilized to provide advance information to the search and rescue efforts by deploying robots to explore the disaster environment before it is safe for the human response team to enter. This permits the response team to develop a more definitive and efficient plan of action for rescuing survivors or shoring up the area to prevent further destruction. However, there are limitations on what robots can accomplish in disaster environments, with communication significantly restricted and the environment essentially unknown. To overcome these obstacles, this research has focused on creating novel algorithms for online, multi-robot exploration to achieve full coverage of an unknown environment using small, basic robots.

The primary contributions in this thesis are two distributed algorithms for exploration using small teams of robots with limited communication. The innovation in these algorithms comes from how the robots disperse into and subsequently explore the environment, even with communication restrictions. We provide theoretical analysis that shows the algorithms will achieve full coverage of the environment, and return all functioning robots to the entry point. The more restricted algorithm uses the minimum number of message types. We demonstrate the algorithms’ functionality using simulations and experiments using physical robots, and compare their performance with other algorithms. The primary algorithm is able to perform on par with other algorithms in spite of the communication restrictions, in multiple environments.
Contents

Acknowledgements i

Dedication iii

Abstract iv

List of Tables viii

List of Figures xii

1 Introduction 1
   1.1 Organization ................................. 5

2 Related Work 6
   2.1 Coverage and Environment Decomposition ............ 8
   2.2 Control Structure ............................... 12
      2.2.1 Centralized Approaches .................. 12
      2.2.2 Distributed Approaches .................. 16
      2.2.3 Discussion ............................... 20
   2.3 Levels of Communication ........................ 21
5.3.6 Simulation Results: Unstructured Environments . . . . . . 96
5.3.7 Physical Robot Experiment Results . . . . . . . . . . . . . 110

6 Conclusions and Future Work 113

6.1 Limitations and Future Work . . . . . . . . . . . . . . . . . . . 115

References 119
List of Tables

3.1 Sweep Exploration Algorithm Transitions ........................................ 45
5.1 Common performance metrics for online coverage algorithms. ........ 81
5.2 Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Simple Office environment using one and three robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots. ........................................ 85
5.3 Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Simple Office environment with three robots. NSS means “not statistically significant.” ........................................ 85
5.4 Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the NHH environment using one and four robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots. ........................................ 86
5.5 Significance \((p)\) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the NHH environment with four robots. NSS means “not statistically significant.” .......................... 87

5.6 Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Hospital Section environment using one, four, eight, and 12 robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots. .................. 90

5.7 Significance \((p)\) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Hospital Section environment with four robots. NSS means “not statistically significant.” .......................... 91

5.8 Significance \((p)\) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Hospital Section environment with eight robots. NSS means “not statistically significant.” .......................... 91

5.9 Significance \((p)\) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Hospital Section environment with 12 robots. NSS means “not statistically significant.” .......................... 92
5.10 Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Cave environment using one, four, and eight robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots. ........................................ 98

5.11 Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Cave environment with four robots. NSS means “not statistically significant.” ........................................ 100

5.12 Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Cave environment with eight robots. NSS means “not statistically significant.” ........................................ 101

5.13 Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Gym environment using one, four, and eight robots. Values in bold denote the best for that metric for a given number of robots. ............... 103

5.14 Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Gym environment with four robots. NSS means ”not statistically significant.” ........................................ 105

5.15 Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Gym environment with eight robots. NSS means ”not statistically significant.” ........................................ 106
5.16 Travel distance in meters, and time to coverage and return in seconds, for each algorithm and number of robots combination run. Values in bold denote best for that metric for a given number of robots.
List of Figures

3.1 Different scenarios where a robot, R1, is in a dead-end. Circles represent robots; squares represent beacons that need to be revisited to complete the exploration; and x’s represent beacons in explored areas. ................................................................. 26

3.2 Different scenarios where a robot, R1, is at an intersection. Circles represent robots. Arrows show open paths from R1. ................. 27

3.3 Finite state machine for robots using the Rolling Dispersion Algorithm. The states are labeled with the initials of the corresponding behavior: AC for Avoid Collisions, D for Dispersion, FP for Follow Path, G for Guard, R for Retract, and SC for Seek Connection. 31

3.4 A small example environment. Circles with identifiers in them are robots. ................................................................. 35

3.5 RDA staggered dispersion method example. Initial dispersion is complete. Robots are circles with identifiers inside and explored beacons are X’s. ................................................................. 36

3.6 RDA staggered dispersion method example. The robots have fully explored the environment. Robots are circles with identifiers inside, unexplored beacons are squares and explored beacons are x’s. .... 37
3.7 RDA staggered dispersion method example. Retraction to location A is complete, and R2 is investigating B. Robots are circles with identifiers inside, unexplored beacons are squares and explored beacons are x’s. ................................. 38

3.8 RDA chained dispersion method example. Maximum dispersion achieved. Robots are circles with identifiers inside, unexplored beacons are squares and explored beacons are x’s. ......................... 40

3.9 RDA chained dispersion method example. Exploration almost complete, a loop and a dead-end have been identified. Robots are circles with identifiers inside, unexplored beacons are squares and explored beacons are x’s. ................................. 40

3.10 RDA chained dispersion method example. Retraction from first path exploration is almost complete, and R5 and R4 have moved out to finish the exploration of the unexplored beacons. Robots are circles with identifiers inside, unexplored beacons are squares and explored beacons are x’s. ................................. 41

3.11 Finite state machine for robots using the Sweep Exploration Algorithm. ................................. 45

3.12 SEA progression example. All robots are out in the environment, and one dead-end has been marked. Robots of any state are circles with identifiers inside, Sentry beacons are squares and Repel beacons are x’s. ................................. 47

3.13 SEA progression example. Exploration is almost complete, and two loops have been blocked (the third loop has not yet been detected). Robots of any state are circles with identifiers inside, Sentry beacons are squares and Repel beacons are x’s. ................................. 48

xiii
3.14 SEA progression example. Exploration is complete, and the robots are in the process of retracting to the entrance. Robots of any state are circles with identifiers inside, Sentry beacons are squares and Repel beacons are x’s.  

4.1 (left) Cave-like environment used in simulations, showing overlay graph. (right) One possible exploration using four robots. The star marks the start, and filled shapes are where dead-ends are detected on a path, initiating the retraction process.  

5.1 The Simple Office environment with exploration partially completed. The four robots are centrally located, and are moving right to left. Two beacons have been activated in the doorways of the rooms on the right.  

5.2 Total distance, average distance per robot, and total time for simulations using one, two, and four robots with each algorithm in the Simple Office environment. Results are averaged over 15 runs for each combination.  

5.3 Average time to full exploration over 10 runs for simulations using one, five and eight robots with each algorithm in the cave environment simulation.  

5.4 Five robots running RDA in the cave environment.  

5.5 (left) Overlay graph of experimental set-up (dotted line denotes start area). The star is the first point in the environment, and the filled circles mark dead-ends. (right) Experimental set-up with cardboard walls. The robots are all located in the start area on the left side.
5.6 Experimental results averaged over five runs with one, two, and three robots using SEA. ................................................. 72
5.7 Experiment environment on the 1st floor of the NHH building. . . 73
5.8 (left) The LEGO EV3 robot used in experiments. (right) Three of the beacons used in the experiments. ............................... 73
5.9 Results of running one, two, and three robot teams using SEA and RDA (there is no difference in results between the two algorithms for one or two robots) in the NHH building. Results are averaged over 10 runs. ....................................................... 74
5.10 Path of two robots at (a) five steps, (b) eight steps, (c) 11 steps, and (d) 15 steps. [19] ................................................................. 77
5.11 Decreasing branching property: worst case scenario. Path of the robots after (a) one step, (b) three steps, (c) six steps, (d) eight steps, (e) ten steps, and (f) 12 steps. [19] ................................. 78
5.12 Boustrophedon paths created by robots $A^1$, $A^2$, and $A^3$. [96] . . 79
5.13 Hospital Section environment used in simulations. ................. 83
5.14 Rate of coverage for the Simple Office environment, using one and three robots. BoB and RDA-MS have identical rates of coverage. RDA and RDA-EC are identical up to 300 seconds, then diverge. 84
5.15 The speedup factor for coverage time with one, two, and four robots with each algorithm in the Simple Office environment. ......... 84
5.16 Rate of coverage graphs for the NHH environment, using one and four robots. Note that there was no difference between the algorithms using a single robot in this environment. BoB and RDA-MS have identical rates of coverage. ................................. 86
5.17 The speedup factor for coverage time with one, two, and four robots with each algorithm in the NHH environment. BoB and RDA-MS have the same speedup factor for each number of robots, so only one line is discernible. ............................................. 87

5.18 Rate of coverage for the Hospital Section environment using one and four robots. ............................................. 88

5.19 Rate of coverage for the Hospital Section environment using one and eight robots. ............................................. 88

5.20 Rate of coverage for the Hospital Section environment using one and 12 robots. ............................................. 89

5.21 The speedup factor for coverage time with one, four, eight and 12 robots with each algorithm in the Hospital Section Environment. ............................................. 89

5.22 Cave environment for simulation experiments. ....................... 97

5.23 Rate of coverage for the Cave environment using one and four robots. 99

5.24 Rate of coverage for the Cave environment using one and eight robots. 99

5.25 The speedup factor for coverage time with one, four, and eight robots with each algorithm in the Cave Environment. .................. 100

5.26 Rate of coverage for the Gym environment using one and four robots. BoB and DFS with one robot have identical rates, and with four robots BoB and RDA-MS also have identical rates of coverage. ............................................. 104

5.27 Rate of coverage for the Gym environment using one and eight robots. BoB and DFS with one robot have identical rates, as do BoB and RDA-MS with eight. RDA’s rate of coverage follows the same step pattern to 100% (not shown). ................................. 104

xvi
5.28 The speedup factor for coverage time with one, four, and eight robots with each algorithm in the Gym Environment. BoB and RDA-MS have identical speed up lines. ............................... 105
5.29 Gym environment with all nodes and possible edges. ................. 106
5.30 BoB exploration path in the Gym environment with a single robot from the starting node (S) to the node where full coverage is achieved (C). ............................................................ 107
5.31 DFS exploration path in the Gym environment with a single robot from the starting node (S) to the node where full coverage is achieved (C). ............................................................ 107
5.32 BoB exploration path in the Gym environment with four robots from the starting nodes (S) to the nodes where full coverage is achieved (C). Nodes covered by a specific robot are connected by arrows depicting the path the robot took, and are separate from the areas covered by other robots. ............................... 108
5.33 RDA-MS exploration path in the Gym environment with four robots from the starting nodes (S) to the nodes where full coverage is achieved (C). Nodes covered by a specific robot are connected by arrows depicting the path the robot took, and are separate from the areas covered by other robots. ............................... 108
5.34 RDA exploration path in the Gym environment with four robots from the starting node (S) to the node where full coverage is achieved (C). Nodes are coded by which robot first reaches that location, and arrows show the order of exploration. ............................... 109
5.35 Results of running one, two, and three Lego EV3 robots using RDA, RDA-EC, and RDA-MS in the NHH environment. ..................... 112
Chapter 1

Introduction

When we think of exploration, we often think of discovering things previously unknown. The Vikings discovered Iceland and crossed the Atlantic to Greenland and North America. Roald Amundsen led expeditions to reach the South and North Poles. Jacques Cousteau explored the underwater environment as a deep sea diver. These all added significantly to our knowledge of the planet, but there are smaller, more mundane explorations as well. We explore when we visit or move to new places. We have robots that explore and then plan a path to vacuum our homes. Exploration can mean discovering something new to an individual, or something new to the world.

Sometimes, the places we need to explore are dangerous, so we have developed robots to go into the dangerous areas and send back the information that we need. The Mars Rover is sending us information about that planet’s surface, and the Voyager mission has explored our Solar System and is moving on to the space between galaxies. Closer to home, the Recon Robotics Scout robot is used by law enforcement and the military to provide reconnaissance before they go into potentially dangerous situations [82]. Robots are also used in the aftermath
of disasters, such as the CMU snake robot which searched the rubble after an earthquake [29], and the Mini-Manbo robot which sent back information about the state of the reactor cores at Fukushima [37]. However, most of the robots in use in these situations are single, teleoperated robots, limiting their range and usefulness in large-scale scenarios [17, 77], such as a hurricane or tsunami. Using multiple robots could allow for quicker exploration of the environment, but presents new challenges when it comes to coordinating the robots.

Large-scale disaster scenarios often come with variable and reduced lines of communication [84]. This is even more true when inside the rubble, where the wreckage itself impedes communication signals. So an important facet of developing robots to aid in search and rescue efforts in the wake of a disaster is to accommodate the limited communication, while exploring the environment. Since search and rescue carries with it the objective of finding survivors or areas of interest, the exploration must also provide complete coverage of the environment. This means that for a robot system to be effective and used, the robots cannot miss a single spot in the environment, because that could be the most critical location for finding survivors. There is a need for algorithms that take these real world challenges into account while maintaining the objective of providing full coverage of an environment which is unknown at the start of the exploration.

There are multiple methods for a team of robots to explore in order to provide coverage of an unknown environment. Gage [44] proposed three categories of coverage: blanket, barrier and sweep. In blanket coverage, the agents provide continuous coverage of every point in the environment. Barrier coverage is often used in surveillance, where a perimeter is set up around an area such that nothing can pass into or out of that area without being seen by at least one agent. Sweep coverage makes a pass over the environment and ensures that every point
has been seen at least once, but the agents don’t stay in any one location, instead moving progressively through the environment. Choset [27] later presented an extensive overview of coverage path planning algorithms according to those categories. Most coverage algorithms are focused on surveillance and usually entail creating a sensor network to provide either blanket or barrier coverage of the environment. Attempting to provide blanket coverage can require a prohibitively large number of robots and some algorithms still don’t achieve full coverage.

This thesis addresses the problem of fully exploring an unknown environment using robots operating under multiple constraints. In the target scenario, blanket coverage isn’t necessary, nor is a static sensor network. Since the environment is unknown, the required number of robots for full coverage is also unknown, and, even if known, may well exceed the number of robots available on site. Therefore, any algorithm we develop must operate correctly with any number of robots, including a scenario where we have only a single robot. The algorithms must also handle the fact that communication between robots in the team, or with the outside world, will be limited in range and bandwidth, and may change as the robots move around. Because we don’t know what robots will be available in a given situation, the algorithms must have minimal requirements for the robots’ movement and sensing abilities, and be generalizable to multiple types of robots.

We have developed and implemented two distributed algorithms that operate under different communication restrictions. The Rolling Dispersion Algorithm (RDA) uses Wi-Fi communication, while the Sweep Exploration Algorithm (SEA) makes use of chemical signals, which greatly reduces the speed of coverage, but provides stronger guarantees about coverage time. By using a distributed method, we are able to take advantage of the redundancy and robust nature afforded by the team of robots. The team size was purposefully kept too small to provide constant
surveillance of the entire environment concurrently. Instead, the robots drop off beacons (such as ZigBee motes or RFID tags) at key locations in the environment to provide longer lasting trails and information to mobile agents that may pass by later. Both algorithms achieve a faster rate of coverage with multiple robots over a single robot.

One of the primary innovations in this thesis was to use the communication restrictions themselves (limitations on range and bandwidth) to direct the robots’ movement. By keeping the robots grouped together during the entire exploration, the algorithms can reduce the likelihood of robots getting lost and the possibility that part of the environment will be overlooked. The other important innovation lies in making the algorithms independent of the type of communication used, while still making the robot team capable of achieving full exploration of the environment in an efficient manner even with some attrition.

We provide theoretical proofs of correctness and completeness for both algorithms, showing that they will achieve full coverage of the reachable area of an environment, while avoiding unnecessary repeated coverage, so long as a single robot remains active until the end of the exploration. We also show that the algorithms work through simulations and experiments with physical robots. Lastly, we provide comparison results with two similar online multi-robot coverage algorithms, though they do not employ communication restrictions, leading us to relax our restrictions to put the algorithms on more even footing. These relaxations are to either start the robots in smaller teams in multiple locations, or to double the communication range. With these two modifications, the Rolling Dispersion Algorithm performs on par or better than the comparison algorithms in terms of rate of coverage and distance traveled.
1.1 Organization

The rest of this thesis is organized as follows:

- Chapter 2 presents previous work in the area of robot exploration, reviewing the advantages and drawbacks to different algorithm facets. This includes types of coverage, environment decomposition, single vs. multi-robot systems, centralized vs. distributed control structures, and varying levels of communication.

- Chapter 3 describes the Rolling Dispersion Algorithm, its modifications, and the Sweep Exploration Algorithm in detail, including examples of how each will explore different types of environments.

- Chapter 4 provides an analysis of the algorithm correctness and properties. This chapter shows that the algorithms will provide complete coverage under the expected conditions, and that the Sweep Exploration Algorithm uses the minimum number of message types to provide this coverage.

- Chapter 5 covers the simulations and experiments that empirically show that these algorithms function as expected. Comparisons to other researchers’ algorithms are included.

- Chapter 6 presents a final discussion of the algorithms presented in this thesis, their limitations, and avenues for future work.
Chapter 2

Related Work

In recent years, multi-robot systems have gained popularity due to decreases in the cost and size of the components made possible by hardware advances [8]. There are several advantages to the use of a multi-robot system over the use of a single robot, including cost, efficiency and robustness [45]. A single robot can be designed to efficiently complete its task, but it may then be suitable for only a small set of tasks. Added functionality increases the cost, size and energy requirements, while reducing maneuverability. In addition, if part of the robot fails, the robot may fail at the entire task. In contrast, a multi-robot system comprised of 10 or 100 smaller, individually less-capable robots, with several of each type needed to complete the different parts of the task, can still accomplish their goal even if some of them fail. The multi-robot system has an inherent redundancy that increases the system’s robustness [27].

A multi-robot system may also be more efficient for some tasks, as the robots can work separately so that multiple parts of the task can be accomplished in parallel. Consider a collapsed building with survivors trapped inside, but the building is still too unstable for human rescue workers to enter. A team of small
robots could be sent in immediately to map out the interior and locate survivors, so that the human rescuers can more quickly reach the key locations when it is safe for them to enter. A single robot would be able to search only one room at a time, while multiple robots could search several rooms at once. Think of Mars, with so much area left to explore, yet still out of reach. A single powerful robot can cover only so much area, but if a hundred smaller robots coordinated to explore further away and relay information to the more powerful robot, much more territory could be covered in the same amount of time. Imagine a wildlife monitoring station, where the humans disrupt the environment and the behavior patterns they wish to observe with their presence. Tiny sensors and unobtrusive collection robots could monitor the area and the humans wouldn’t need to enter. A team of robots could be sent to clean up a hazardous waste spill, or to de-mine a battlefield, or act as surveillance in a critical area. More mundanely, robot teams could be deployed to clean leaves and other debris from city streets, or mow lawns or vacuum floors. Teams of smaller robots can be more effective than a single robot in such scenarios [22, 27, 45], but there are still many issues facing them, such as coordination, connectivity and deployment management to direct efficient movement of the individual robots.

In this chapter we review previous research in this area, including the types of coverage and how the environment is broken down for the exploration, the control structures for the team, including hierarchy and centralized vs. distributed control, and how different levels of communication affect the exploration. We discuss costs and benefits of each system, including feasibility for real world application, since many of the existing systems have been tested only in simulation. This previous work has helped to direct the design and development of the algorithms presented in this thesis.
2.1 Coverage and Environment Decomposition

The first question to consider for an exploration system is what type of coverage does the system need, and how will it break the environment down to measure its performance. Gage’s [44] three types of coverage are a good starting point, and the application of the exploration system can help determine which type should be used. If the task is to provide security camera coverage for a museum, this can be exactly calculated for the museum’s layout and then cameras can be acquired to provide the needed blanket coverage. Alternatively, if the objective is to detect when cattle break through the enclosing fence, the system should be set up for barrier coverage, and again, we can exactly calculate the necessary number of sensors to provide that. Both blanket and barrier coverage require a set number of agents—robots or sensors—for a particular environment, and those numbers can vary widely for different environments. This means that it is easiest to provide these types of coverage for applications where we can calculate the needed number of agents in advance, and where the environment is unlikely to change drastically.

In contrast, sweep coverage is a flexible way of exploring every accessible part of an environment with only a few mobile agents. While blanket and barrier coverage systems can be set up in advance, and thus can use stationary agents like cameras or motion sensors, sweep coverage requires the agents to be capable of moving through the environment, to provide the full coverage that is requested of the system [3, 21].

In addition to the type of coverage, a system needs a way to break down the environment in order to measure its progress towards full coverage. When talking about blanket coverage, one usually uses Voronoi cells or triangular decomposition to distribute the agents for coverage. A triangular distribution is provably
complete, with agents placed at the corners of the triangles; Ma and Yang [70] show that this provides an adaptive distribution for mobile sensor networks. Both Voronoi and triangular methods can be used to create graphs or trees of the environment as well, and then a search method can be applied to provide paths for full coverage (if the environment is known in advance), or the tree/graph can be built up as the agents explore, resulting in a connected network for future passes. Another form of decomposing the environment is cellular decomposition, which divides the environment into a grid of equal sized cells. A cell is considered covered if an agent has passed through it (for sweep coverage) or one or more agents’ sensors can observe the entire cell (for blanket and barrier coverage). Viet et al. [96] developed the BoB algorithm, which decomposes the environment in this way, and then completes a sweep coverage of the environment, with each robot working its way through each individual cell of the environment. Grid-based approaches can be quite effective in ensuring full coverage of an environment, and are particularly well suited to smaller, indoor spaces [45], but the size of the grid cells can require longer travel distances to achieve full coverage. In contrast, a graph based approach allows the robots more latitude in choosing where to explore [19, 62], and decomposes the environment in a way that is more tuned to that particular environment, but with less structure, there can also be more overlap in the robots’ sensor coverage. Brass et al. [19] are able to provide optimal paths for two robots using their algorithms for exploration of undirected graphs, particularly environments such as office buildings or caves.

When a system for exploration of unknown environments attempts to provide blanket coverage, it usually has the goal of maximizing coverage, but not necessarily providing full coverage. In a task where full coverage is not required, this is an interesting problem, from the perspective of dispersing the agents into the
environment. Kataros and Zavlanos [57] present a method for distributing their mobile sensors in an environment to provide maximal coverage while maintaining reliable communication between the agents. They use Voronoi cells to define the covered area of each agent as they move into the environment. Similarly, Ludwig and Gini [69] developed a system that uses the agents communication to disperse them for maximal coverage in an unknown environment, with simulation results for up to 100 robots in a very large and complex environment. Rooker and Birk [84] also developed a system to provide maximal coverage, while maintaining communication, using simulations and limited numbers of robots. The need for so many robots is one of the biggest drawbacks to using blanket coverage in real world online exploration tasks. When the environment size or structure is unknown, being able to bring the correct number of robots becomes challenging, and very few places can supply the number of robots needed for blanket coverage in large environments. Environment structure matters because the coverage could be measured in terms of line of sight, or communication range, or sensor range, all of which depend on where walls and other obstacles exist in the environment. Since our main focus is on disaster scenarios, we anticipate that the environment structure will not be known, and the environments will frequently require 50 or more robots to provide blanket coverage, which is not a reasonable number to expect local rescue responders to have on hand.

One way that some researchers resolve this is to provide persistent coverage, in which the agents have a designated area to cover, but they have to repeatedly move around that area to provide the coverage, because the assigned area is too large for each robot to provide simultaneous coverage [75, 95, 98]. This can lead to a need to provide refueling locations for the agents, because the persistent coverage costs more in terms of energy [33, 65, 74], and that can make it difficult
to provide guarantees that the robots won’t leave an area uncovered for too long between visits [1, 39]. This can also apply to barrier coverage, where a perimeter is too large to provide full coverage simultaneously, but agents can move back and forth along a segment to keep it under observation [2, 35, 53]. Persistent coverage, both blanket and barrier, is also widely used in adversarial situations, in which the agents are trying to detect or prevent opponents from getting through the environment [40, 78, 81, 101]. One can argue that persistent coverage should be associated with sweep coverage instead of blanket or barrier, but often the goal of sweep coverage is to make a single pass through the environment, rather than repeated passes. This is not always the case, but the line between the types of coverage becomes more blurred when persistent or repeated coverage comes into play.

Blanket and barrier coverage are popular for multi-robot systems, because why not provide simultaneous full coverage if we can? But many of these systems were only tested in simulation, and if we try to take them into the real world, the necessary number of robots makes the systems infeasible for common use. On the other hand, sweep coverage is often associated with single robot systems, because it is the only type of coverage that a single robot can provide—making its way through the environment, but only able to provide coverage for its sensor range at any given time [27]. Limiting the number of robots in a multi-robot system, though, makes sweep coverage appealing again [41]. Low et al. [68], Parti [80] and Viet et al. [96] also implement sweep coverage for their exploration systems in unknown environments.

Having reviewed these options, we decided that sweep exploration was more in line with our objectives, especially the fact that we cannot guarantee a sufficient number of robots for real world situations. We also chose to use a tree-based
approach to help provide flexibility in how the robots disperse. The robots build the tree as they explore, because the environment is initially unknown, so the resulting tree is based on the communication signal intensity between agents at the nodes, rather than using either triangular or Voronoi decompositions [54, 56].

2.2 Control Structure

Determining the coverage and decomposition methods then leads us to how the team of robots is controlled and coordinated. This includes the structure of the team, from strict hierarchical role structures that are common in search and rescue procedure, to the loose and interchangeable roles and leadership that one can see in sports such as ultimate frisbee [30], where the players switch positions and who makes the decisions based on who has control of the frisbee and who their closest neighbors are. Coordination is also broken down to centralized, decentralized and distributed systems. We are primarily interested in centralized vs. distributed systems in this work. The effectiveness with which the robots cooperate is critical to the viability of using them for search and rescue efforts [77], and can be based in human communication structures [11], but needs to also assume different types of communication and coordination.

2.2.1 Centralized Approaches

In a centralized approach, either a small set of the robots or an outside machine issue instructions to the others and keep the group organized and coordinated. This can be effective because it keeps different robots from covering the same area multiple times. It also means the individual robots require less computational power, though the central controller needs more computational ability. However,
this also makes a single point of failure for the system, reducing the benefit of having multiple robots, because they will not function without the central controller. In addition, a centralized approach does not scale as well, because one machine can efficiently control only a limited number of robots at a time. This also reduces the distance the robots can move from the central controller because they must maintain connectivity.

In small unknown environments, though, a centralized approach can be effective. Stump et al. [92], made a single robot the base station, which directed the other robots to maintain a communication bridge as they extended into the unknown area. Maintaining connectivity allowed the robots to explore further by keeping a trail back to the base station, and also made it possible for a global map to be built and stored centrally. Similarly, Rekleitis et al. [83] used one robot as a stationary beacon for another robot, thus reducing odometry error in the robot that was moving through the environment. If expanded to a larger group of robots, such an approach would make it easier to merge maps between robots, and also make the maps more reliable as there would be fewer errors. These approaches, where the range is limited by communication constraints, require constant monitoring of the individual robots in order to keep the map consistent and the exploration efficient.

In Stachniss and Burgard [91], the use of a global coverage map to maintain the exploration status of each cell aides the centralized system in directing robots to unexplored areas of the environment. Burgard et al. [20] used a central monitor to detect areas of interest as they are located and instruct robots to explore these targets. When a robot approaches an area, the utility of sending another robot to a nearby location is reduced. This keeps the robots more dispersed and leads to greater coverage with little overlap. Similarly, Wurm et al. [99] used a
central computer to identify frontiers, and deploy robots to segmented areas off the frontiers. This approach is less intensive for the central controller, because the individual robots can complete the exploration of their segment before reporting back. However, this is a slower exploration process because additional robots cannot be deployed until new frontiers are located. This leads to few robots exploring at the start, which may not be an efficient use of resources.

Hazon et al. [47, 48] extended their offline multiple spanning tree coverage algorithm for use in online situations. They proved in simulation that their algorithm would provide complete coverage, even in the face of individual robot failures, if at least one robot remained active until the exploration was complete. However, they made several assumptions that limit the use of their algorithm in real-world scenarios. The assumptions that the robots can be given their exact starting location, have unlimited communication range (for connectivity to the central manager), and can localize themselves anywhere in the environment are acceptable in simulations, but are very infrequently practical in physical implementations. Another study applied team formation to the robots, allowing them to split at decision points, but assuming that the robots would create perfect maps individually that would be trivial to splice together when the robots regrouped [61]. This can be done in simulation, but is very difficult to achieve in practice. These approaches require constant monitoring of the individual robots in order to keep the map consistent and the exploration efficient [24, 25, 87] Further, the range is limited by communication restraints.

Two further studies did not require their robot teams to cover the entire environment, but rather cover certain the target locations. In Low et al. [68], the objective of the team was to locate and extensively explore points of interest, but while taking much more infrequent readings in the regions between targets. This
adaptive sampling method expends less energy over the course of the exploration and allows the robots to cover a larger area. Similarly, a market-based approach allows a central manager to define tasks and then allow the individual robots to bid on the tasks [102]. The central controller can accept the lowest bid without having to know much about the bidder, reducing the computational expense on the side of the central controller. This approach could be applied to some of the previous studies, such as Wurm et al.’s segmentation method [99], where a robot could, instead of being instructed to move on, bid on various frontiers near its current segment.

In all of these centralized cases, the majority of the computational expense is placed on the central controller. In tasks with only a few robots, or a limited area to explore, this can be effective, and the centralized aspect of building the map is convenient for human operators. For a small building collapse, a centralized view is more useful, because the human rescuers need a full map of the area, and must keep contact with all of the robots. On Mars, it is prohibitively expensive to equip an entire team with long range communication hardware for sending data back to Earth, so the less capable robots must update the more powerful one with their sensor readings. However, the messages sent back to Earth aren’t constant, and the smaller robots are more efficient if they have some autonomy to explore without supervision. Likewise, in a large scale disaster area, no single human operator, or even single control center, could keep up with the number of robots needed for the search and rescue mission. In these cases, a distributed system is a better option.
2.2.2 Distributed Approaches

A distributed coverage algorithm scales much better than a centralized one, and can also take full advantage of the robustness inherent in the redundant nature of a multi-robot system. Many of the distributed methods for coverage come out of the sensor networks field. Though much sensor network research uses pre-deployed sensor nodes, and the environment is known \textit{a priori}, there has also been some work in unknown or dynamic environments applied to mobile sensor networks. Ma and Yang [70] show that the most efficient dispersion of mobile nodes is triangular, producing the maximal overall coverage and minimal overlap or gap in the coverage. The dispersion formation is achieved through the nodes’ local communication, in which they determine distance and bearing to their neighbors, so that they can move towards the optimal formation. Sensor nodes frequently have very limited power and thus can move only short distances, but this work shows that it is possible for the nodes to self-configure. Though many mobile sensor network studies aim to reconfigure a network once and then let it remain static, Liu et al. [66, 67] have shown that repeated location updates can lead to better coverage over time. These approaches can easily be applied to robots as well, especially given that robots generally have much longer battery lives and more computational and sensing ability than sensors. Similar approaches by Howard et al. [50, 51] and Cortes et al. [31] used potential fields and gradient descent, respectively, to disperse the nodes. In simulation, both methods effectively spread the nodes throughout the environment.

There is also interest in modeling distributed algorithms for multi-robot systems on insect behavior [9]. There are two approaches: swarm- and pheromone-based systems. Swarms, based on insects like bees, use direct robot-to-robot communication to coordinate the movements of all the individuals. The robots
have very little ability themselves, but can communicate with local neighbors and use simple distributed algorithms to arrange themselves according to a desired dispersion pattern. McLurkin and Smith [72, 73] have developed both a physical robot and several algorithms for both dispersion and exploration in indoor environments. Their algorithms rely on the robots maintaining connectivity in order to perform correctly, passing information amongst themselves to spread out in particular patterns, such as uniform and cluster dispersions, as well as maintaining a frontier for exploration. The robots keep lists of their neighboring robots, and use gradients based on the network of robots to direct their movement. These algorithms are similar to those in [31, 51, 70], but allow for greater variability in the dispersion pattern, including clusters and perimeter formations. The robots can also perform tasks such as frontier exploration and following-the-leader, which is not considered in sensor network research.

The second of the insect-based approaches imitates ant behavior and uses pheromones as a passive communication vehicle amongst the robots. Koenig and Liu [59] conducted a series of simulations on the efficacy of several algorithms for pheromone-based coverage. They showed that allowing the robots to share markings (by reading those left behind by other robots that had previously covered the area) led to faster coverage of the environment. Because it was a simulation study, they were able to use simple counts of each visit to a cell in the environment as pheromone traces, which is not as feasible in real-world implementations, though Sugawara and Sano [93] did build a system in which overhead cameras tracked the robots to maintain the pheromone trails. Mamei and Zambonelli [71] used RFID tags to store pheromone traces as objects and robots moved through the environment. An object would leave an ascending count in the tags along its path, and a robot could later follow the gradient uphill in order to find the object’s current
location. Though not focused on coverage, this study does highlight one of the main limitations in pheromone-based systems—that it is difficult and expensive to pervasively leave traces that a robot can make use of in a physical environment [43, 45, 49]. If the devices must be placed in the environment beforehand to be useful, then the environment is not unknown, which defeats the purpose of exploring it. However, if the robots were to leave the devices as they explored, then the nodes become a useful aid to continuous or repeated coverage of an environment. In [16], Batalin and Sukhatme have the robots store what they know of the environment on a node before dropping it off, and the node can be updated by robots that pass later, as well as offer suggestions of paths least recently taken from that point. This type of hybrid multi-robot and sensor network system can be more effective than either by itself. A similar experiment was conducted by O’Hara et al. [79], though it makes use of pre-deployed sensor nodes. In addition, it the current implementation, the necessary density of the sensor nodes prevents deployment in a high-traffic environment, because the nodes would be moved or destroyed by those moving through the area.

An alternative approach to these swarms is to allow the robots to separate in order to explore distant frontiers, and work towards each other to build a map of the environment [4]. This allows a single sweep through the environment, but also relies on precise measurements from the robots’ sensors to fuse the maps together into a global whole.

There is also a great deal of work in the sensor network and control theory fields. Dirafzoon et al. [34] provide an overview of many sensor network coverage algorithms, both centralized and distributed, which can be applied to multi-robot systems as well. However, many of these rely on individual robots knowing the
distance and bearing of other robots around them, which requires more sophisticated sensors and defeats some of the purpose of using a team of basic robots. For example, Kurazume and Hirose [60] developed an algorithm in which the team of robots was split into two groups, one of which remained stationary while the other moved, and then they traded roles. This made for effective movement through an unknown environment, but the robots relied on sophisticated sensors to perform dead reckoning to determine the locations of the stationary robots. At the other end of the spectrum, there is research that has shown that a team of robots can disperse into an unknown environment using only wireless signal intensity to guide the dispersion [54, 69]. This method allows the use of small, simple robots, without the need to carry a heavy payload of sensors, so that the robots can run longer and explore further. Smaller, simpler robots are less expensive, so more robots can be acquired for the same task. Overall, a distributed system allows an individual robot to work independently, without need for long distance communication, while also sharing data with neighbors as necessary.

Distributed systems can be deployed over a large area, and are more resilient to the failure of a single robot. Instead of a central controller issuing commands and collating data, each robot is responsible only for its own movements and data collection, and relies only on local neighbors for coordinating their exploration and dispersion. This spreads through the entire group, so that it may seem that they are working together on a global scale, but in actuality the decisions are made individually on a local scale. Information can be passed throughout the group, similar to the communication bridge in [92], but it is more of a broadcast message than a directed message along a single path. On a city-wide scale, a search and rescue team does not need information about the entire city, but only the few blocks under its supervision. A distributed system would allow a local group to
work independently, while sharing data with neighboring groups as necessary.

2.2.3 Discussion

Both centralized and distributed multi-robot systems have their strengths, and can be better for specific scenarios. They can also be combined, as in the case of Mars exploration, taking the necessary pieces of each to make a system appropriate to the situation. However, there are still significant limitations in much of the previous research.

McLurkin and Smith’s swarm study [72] is one of the few that uses physical robots, and on a larger scale than most such studies. In fact, most of the experiments conducted in real-world scenarios are from the sensor network field, while much of the multi-robot studies have been implemented in simulation only. Though the algorithms work in simulation, it is highly unlikely they would be able to be ported directly to physical robots. Most simulations do not model effects such as gravity or the variability in a sensor’s readings (which can range widely even for the exact same location with the robot in the same orientation). Simulations also make placing pheromones, as in [16] or devices in the environment much easier. Real-world implementations are much slower than simulations [79], making it more difficult to run sufficient tests to statistically ensure completeness or correctness of an algorithm. Additionally, simulations need not worry about battery life or limited computational capability on individual robots, which can have a great impact on the robots’ ability to complete a task.

Many of the physical robot experiments have come from two areas: single robot systems and sensor networks. Though the techniques and algorithms can be applied to multi-robot systems on their own, it may prove more beneficial to use hybrid systems, such as those in [16, 71, 79], in which a sensor network
supports the movement of the robot team by acting as beacons or information repositories. The sensor nodes require less energy, especially if the connectivity requirement is removed. Without connectivity, the sensor nodes become useful only on a local level, while the robots must move throughout the environment to construct the global view of the area. A hybrid system of robots and sensor nodes also makes more sense in a habitat monitoring scenario, because the static sensor nodes won’t disturb the wildlife, and a robot that comes by to collect the data can be less disruptive than a human.

In our algorithms, we took the approach of using a distributed system, with the addition of beacons placed in the environment to build a network that can help guide the robots to frontiers and to the entrance. Another benefit of the beacons is that the human search and rescue team can use them later to more easily find the points of interest in the environment. While not as good as a map in some cases, the network requires significantly less computational and sensor capability of the robots and sensors, which we feel makes it easier to implement for real world scenarios.

2.3 Levels of Communication

There are already some robotic systems in use for search and rescue or disaster relief scenarios. CMU used their snake robot to help with search efforts in Mexico in 2016 [29], the Mini-Manbo underwater robot was used to find the remaining cores from Fukushima [37]. Robots were used in the aftermath of the World Trade Center disaster [23, 64, 76]. One of the limitations of these systems is that they were all single robots. With the exception of the Mini-Manbo, they were also all tethered and teleoperated, which restricted their usefulness and effective
exploration range. With the limited modes and bandwidth of communication during disaster scenarios [45, 77], this can make the rescue workers wary of using robots for the search efforts, particularly robots that operate autonomously and without direct human oversight [88, 89, 94]. There are also systems that explore in cooperation with human operators, but managing multiple robots can be difficult for a single operator [6, 42]. Rosenfeld et al. [85] created a system to help manage that information flow, giving the robots some autonomy, but also providing the operator with pertinent information. Andolina and Forlizzi [6] developed a tablet-based system for human operators to control robots, and are using it to study how humans control and interact with the robots during operation. Chen and Barnes [26] discuss human factors and the issues they raise when humans are tasked with controlling multi-robot teams. In all of these cases, the content and quantity of communication messages can vary, but must exist to at least a small extent. The level of communication needed by a system in order to complete the exploration is an important factor in making a system that will be used.

Many existing systems assume global communication, particularly centralized systems, though some distributed systems also require it [96]. However, having global communication between all robots and all human operators at all times is an impractical goal. Instead we want to take the communication limitations into account, and build systems that limit the amount of communication necessary for accurate functionality. This can range from no communication between agents at all, as in Kernbach’s swarm approach [58], to communication at meeting locations, as in Brass’s approach to having the robots communicate only when at intersections [19], to requiring that the agents maintain connections with at least one other agent, as in Kataros’ reliable communication for providing maximal coverage with a multi-robot team [57]. Maintaining communication can also allow
for providing coverage guarantees, and even coverage time guarantees [90]. Most swarm approaches tend towards no direct communication between agents, while systems that create maps or networks rely heavily on communication at least at the local level [12, 97]. Reduced communication may be handled by altering how the robots make decisions, if there is time to decide on a policy, which Amato et al. [5] sped up using finite state machines to encode the policies. Bradshaw et al. [18] also explore choosing adjustable levels of autonomy for control policies.

When the robots do not need to be constantly in communication, they can achieve separate tasks, but may occasionally need to check in with each other. Meeting or communicating at set times, or coordinating asynchronous communication is one approach [13], and creating communication maps, to provide reliable way points for updating the group is another [10, 14]. Handling recurring communication needs, so that the robots can connect to a base station when new information is discovered, or to learn if other robots have found new information is the focus of Banfi et al.’s multi-robot exploration system [15]. It may also be that robots need to coordinate to reach a particular area, in which case they need to be able to communicate that need, but without always maintaining contact, so as to be able to work further apart [7].

In our algorithms, we opted to require that the agents maintain communication with at least one neighbor, and we tied that communication into how the robots decide which way to move as they explore. We determined that maintaining communication would help us to provide stronger guarantees of completeness in the exploration, which we consider an essential aspect of developing a system for disaster scenarios. We did consider the amount of communication required, in terms of bandwidth, and tested the effect that had on the exploration path and time.
Chapter 3

Communication-Restricted

Exploration Algorithms

In our algorithms, we wish to achieve full coverage—every point in the environment has been viewed by a robot at least once, so that nothing is overlooked—but we wish to do it with a small team of basic robots. This excludes blanket coverage, due to team size; and also rules out dead reckoning methods for determining robot locations, due to limited sensor and computational capability. We have also chosen a distributed method so that we can take advantage of the redundancy and robust nature of a team of robots. Lastly, as in insect-based algorithms and mobile sensor networks, the robots distribute beacons (such as ZigBee motes or RFID tags) at key locations in the environment to provide longer lasting trails and information to mobile agents that may pass by later.

Keeping the above items in mind in developing the new algorithms, we established the following two restrictions on the system:

- There are not enough robots to provide blanket coverage of the environment.
• Communication is limited in terms of range—distance between communicating agents—and bandwidth—amount of information contained in each message.

We also made the following assumptions:

• The robots have some form of proximity sensors to allow them to avoid collisions while they move about the environment.

• The robots can communicate with each other in some way (wi-fi, line-of-sight, chemical, etc).

• The robots have the means to carry and drop off beacons.

• We are dealing with a disaster scenario, so the specifics of the current environment are unknown, even if information (such as a map) for the pre-disaster environment is available.

Our algorithms use the communication signal intensity to direct the robots’ movements, keeping them linked as a group during the entire exploration. This provides the benefit of reducing both the likelihood of robots getting lost and the possibility that part of the environment will be overlooked. Our innovation lies in making the algorithms independent of the type of communication used, while still making the robot team capable of achieving full exploration of the environment in an efficient manner even with some attrition.

3.1 Common Features

We have implemented two new algorithms, which operate under different sets of communication restrictions, in both simulation and on physical robots. The
algorithms share a number of features, which we describe here, before going into the details of each individual algorithm.

During the dispersion and exploration of the environment, each robot will execute the algorithm and make decisions as an individual, but it will have input from the neighboring beacons and other robots. Beacons are dropped by the robots for three reasons. The first is to mark a path that has been fully explored, thus preventing multiple explorations of the same area. This process is started when a robot encounters a dead-end (see Definition 1), and continues until all robots along that path have pulled back to the nearest intersection with unexplored paths (see Definition 2). The second reason robots drop beacons is to mark the path to an unexplored area, when they need to temporarily abandon that path in order to complete exploration in another area first. The third reason is to mark the path to the entry/exit, so that the robots can leave the environment when the exploration is complete.

**Definition 1.** *An exploring robot is in a dead-end when every direction in which it might move is towards an obstacle, be it a wall, a beacon, or another robot. Figure 3.1 shows several examples of dead-end scenarios.*

![Figure 3.1: Different scenarios where a robot, R1, is in a dead-end. Circles represent robots; squares represent beacons that need to be revisited to complete the exploration; and x’s represent beacons in explored areas.](image)
Definition 2. Intersections are defined as locations where either there is an obstacle creating multiple paths (such as a corner), or where the robot could move in perpendicular directions without obstruction. Figure 3.2 shows several examples of intersections.

![Figure 3.2](image)

Figure 3.2: Different scenarios where a robot, R1, is at an intersection. Circles represent robots. Arrows show open paths from R1.

The robots use the communication signal not only to exchange information, but also to direct their movement, both in dispersing to explore a larger area and to return to the starting point. Wireless signal intensity, used in one of the algorithms, can fluctuate due to obstacles between robots, and may not be the same at every point a set distance from the origin, but this is not critical to the operation of our algorithm. The other algorithm utilizes communication methods that are persistent in the environment, such as chemical dispersion or line-of-sight camera and colored LEDs. In both algorithms, the goal is to maintain communication, so the robots only need to know if the signal intensity is increasing or decreasing to inform their decision on which direction to move. This may not lead to the maximal dispersion, but suboptimal dispersion is acceptable since our main priority is to achieve full coverage without loss of communication.

Our algorithms use signal intensity to disperse the robots, and beacons to mark locations, but the innovation in our approach lies in the manner in which
the robots continue the exploration past the bounds of their initial dispersion. The robots are not allowed to move in isolation, but must always stay within communication range of the team. The highest priority of a robot that has lost communication with the team is to reestablish that communication. When there is an area to be explored that is beyond the reach of the robots nearby, because they would have to move out of communication range to reach it, the entire team of robots will move towards the unexplored area. This approach has two main benefits. First, the robots are less likely to get lost, since they will have a wireless signal to follow to get back to the entrance. Second, the robots will clear each room and corridor in a methodical manner, similar to the pattern used in law enforcement and search and rescue efforts [89, 23], thus reducing the likelihood of missing an area.

While both algorithms share these features, they are otherwise quite different, which makes them suited for different types of problems. We provide the details of their operation, highlighting their differences, and discuss their properties and guarantees below.

### 3.2 Rolling Dispersion Algorithm

In the Rolling Dispersion Algorithm (outlined in Algorithm 1 and shown as a finite state machine in Figure 3.3), each robot uses information about connectivity with its neighbors and nearby obstacles to choose which of the following six behaviors it will execute on each iteration of the algorithm.

**Avoid Collisions:** Use the proximity sensors to avoid colliding with walls, objects, and other robots.

**Disperse:** Move towards open space, checking wireless signal intensity between
myself and my sentry. If wireless signal intensity is not decreasing, change
direction and continue moving forward. Move away from other robots and
all beacons, particularly those marking explored areas.

**FOLLOW PATH:** Alert neighbors that I can fulfill the request. Concatenate my
path with that of the requester at the common intersection, and follow it to
the requesting robot.

**GUARD:** Stay in place and act as a sentry for other robots.

**RETRACT:** Drop a beacon to mark the explored area and return to my sentry’s
location by following the wireless signal intensity.

**SEEK CONNECTION:** First go in reverse a short distance to see if a connection
can be reestablished quickly (found a dead zone or just barely moved out
of range). If that doesn’t work, turn around and move forward, changing
direction occasionally until a connection with another robot is made.

At any given time in the exploration, each robot has a status of either *sentry*
or *explorer*, and it can switch between these roles as needed. The sentries provide
the backbone of the communication network and do not move while in that role.
Explorers move away from sentries guided by the wireless signal intensity, with
some directional input from their proximity sensors. An explorer becomes a sentry
when it reaches the edge of its sentry’s wireless range and there are no other
sentries or explorers that it can use to stay connected to the group. A sentry can
become an explorer under three conditions. One, if the path it was marking has
been fully explored and all robots beyond it have returned past it, then it follows
a path to an unexplored area and continues on as an explorer there. Two, if there
are no explorers left, a sentry marking an unexplored path with no dependent
Algorithm 1 Rolling Dispersion Algorithm: Robot Loop

1: Update connectivity graph using signal intensities and share with neighbors
2: Check for open paths, and update branch_count
3: if I am too close to an obstacle then
4:    set behavior to AVOID COLLISIONS
5: else if I am disconnected from all neighbors then
6:    set behavior to SEEK CONNECTION
7: else if I am in a dead end then
8:    drop a beacon set to explored
9:    set behavior to RETRACT
10: else if my sentry’s intensity is below threshold then
11:    change status to sentry
12:    set behavior to GUARD
13: if my only neighbor is my sentry then
14:    request additional explorers
15: else if I am an explorer approaching a beacon then
16:    if the beacon is marking an explored area then
17:        turn before continuing on
18:    set behavior to DISPERSE
19: else if I have received a request then
20:    set behavior to FOLLOW PATH
21: if I am an explorer then
22:    drop a beacon set to unexplored
23: else if I am a sentry then
24:    if my only neighbor is my sentry then
25:        if my branch_count is lower OR my path length is shorter then
26:            drop a beacon set to unexplored
27:        change status to explorer
28: else if I am at the entrance then
29:    drop a beacon set to entry
30:    change status to explorer
31: else
32:    pass the request on to my neighbors
33:    set behavior to GUARD (overrides previous setting)
34: else if I have reached the requesting robot then
35:    set behavior to DISPERSE
36: else if I am an explorer then
37:    set behavior to DISPERSE
38: else set behavior to GUARD
explorer robots will drop off an *unexplored* beacon and become an explorer and explore a different path; the robots will return to the unexplored path it was marking at a later time. Three, if there are no explorers left, the sentry closest to the entry, with only one child sentry will become an explorer, leaving behind a beacon to mark the path to the entrance. A robot sentry between two robot sentries will never respond to a call for additional explorers.

All robots maintain an up-to-date connectivity graph, which is comprised of all the robots and beacons that are within wireless range of the individual robot, as well as a list of robots or beacons that have become disconnected from the individual robot. This connectivity graph is shared with the robot’s neighbors on each iteration of the algorithm, and the information may be used in the robot’s decision to move in a certain direction. Each robot also stores its path back
to the entry (consisting of sentry and beacon IDs), its sentry, and its current \textit{branch\_count}, which is the number of open pathways from the robot’s current location.

As the robots move through the environment, they establish a path of sentries and beacons back to the entrance. This path can also be used to guide a robot to the edge of the explored area, which is the rolling aspect of the dispersion. Robots retract along their path when they have completed the exploration of that branch. During the retraction step, beacons are left at all intersections to mark the explored area to prevent multiple explorations of the same area. These beacons have a status of \textit{explored}, which they send to any approaching robot to let the robots know not to go in that direction. When a robot has retracted to the beginning of its path, it then moves out along another path, which may already have some robots exploring it, or it may be a completely unexplored path. When all the robots have become sentries, but there are still areas to be explored, a sentry with no dependent sentries will drop a beacon with the status of \textit{unexplored} and move to explore another path. When there are not enough robots to both maintain the path to the entrance and continue the exploration, the sentry at the entrance will drop a beacon to mark the path, and then move down the path to the unexplored area. This beacon will have a status of \textit{entry}, to differentiate it from the other beacons. However, other sentries along the path to the entrance will drop beacons set to \textit{unexplored} if they are needed as explorers elsewhere. A beacon’s status can be changed from \textit{unexplored} to \textit{explored} when the robots retract past it after exploring the area.

In summary, the Rolling Dispersion Algorithm works by first having the robots disperse, and then shifts to a Depth-First-Search approach as needed. The dispersion and exploration movement is primarily directed by the wireless signal
intensities between the robots, as in Ludwig and Gini’s [69] work, though the robots’ direction is also influenced by the proximity sensors, to avoid collisions. Once the robots have reached the maximum dispersion coverage without losing communication with at least one other robot, the majority of the robots will stay in place, while a few leave their frontier and move along another path to complete the exploration of that path. When a path has been fully explored, those robots will retract until they reach a robot that marks an intersection with unexplored paths, dropping beacons as appropriate, and then continue along one of the unexplored paths to complete the exploration. When there are no more paths to explore (i.e. every point in the environment has been covered at least once), the robots will retract to the entry.

3.2.1 Algorithm Example

We provide here two detailed examples of how the algorithm works, first using the original, staggered dispersion method, and then using the improved, chained dispersion method. Each dispersion method is described below, before detailing the robots’ progression through the environment of Figure 3.4. In the example figures, we will label locations where the robots are likely to stop with letters, and doorways will be labeled with numbers, for ease of describing the robots’ progression through the environment. We will use six robots in the examples, identifying them as R1 through R6. In practice, the robots are unlikely to stop exactly at a doorway, or in the center of the corridor or room. However, this example is intended to demonstrate the basic concepts of the algorithm, so for ease of illustration, we have laid it out on a grid system. The algorithm would have the robots stop at the limits of the communication range, no matter where that lined up with various locations and doorways. While the algorithm has the
robots disperse in semi-random directions as they move apart, in a structured environment such as this one, the exploration path is usually the same from one run to the next.

**Staggered Dispersion Method**

Originally, we designed the dispersion method to have a front line of robots that moved away from the start location, one along each path. The remaining robots wait at the entry point until the front line explorers reach branches or the edge of the communication range and send requests for additional explorers. If robots on multiple paths request more explorer robots at the same time, and the total number of branches across requests is greater than or equal to the number of robots at the entry (not counting the sentry at that location), then a robot would be sent down the path with the fewest branches, to see if that path could be completed, and additional robots would be sent that way as well, if needed. This method was developed to avoid having too many robots moving simultaneously in crowded spaces, where they would interfere with each other’s progress. However, this method also proved to be a bottleneck in dispersing the robots and led to some paths being explored only as far as a single robot could go, and then being left until later. In an environment in which there is a single corridor and small rooms off each side (requiring only one robot to complete the coverage of each room), this works well, but with longer paths off each intersection, it was not as efficient as we wanted, as the following example demonstrates.

The robots start in a group at location A, as shown in Figure 3.4. R1 will act as the first sentry and stay at the entry. There are two potential paths from there—through doorway 1 or to location B. R2 will move to location E and stay there as a sentry, with one open path (location F). At the same time, R3 will
move to location $B$ and stay there, with two open paths (doorway 2 and location $C$). Both robots will call for additional explorers. Since R2 has fewer open paths, R4 will move to location $E$ and then on to location $F$. From there, R4 will be able to connect to R3, and, since they have different sentries, R4 will move towards doorway 2 and leave a beacon set to explored, and then move back to location $F$. Now both R3, at location $B$, and R4, at location $F$ have only one open path each. Both will request additional explorers, but R4 has the longer path. Therefore, R5 will move out to location $F$ and then on to location $J$, where it has two open paths (locations $I$ and $K$). R6 will follow R5’s path and then move on to location $I$, where it finds itself in a dead end. R6 leaves a beacon set to explored and returns to location $J$, where R5 has updated its branch_count to 1. R6 then moves to location $K$, where it has two open paths (doorway 6 and location $L$). Figure 3.5 shows the dispersion at this point.

A request for more explorers reveals that there are no explorers left. Since
Figure 3.5: RDA staggered dispersion method example. Initial dispersion is complete. Robots are circles with identifiers inside and \textit{explored} beacons are X’s.

R3 is at the end of its explored path, and has a \textit{branch\_count} of 1, which is less than R6’s \textit{branch\_count}. It drops a beacon set to \textit{unexplored} and follows the path to R6 (A-E-F-J-K). R3 continues on to location G where it has one open path (doorway 3). Again, there are no free explorers, so R1 drops a beacon set to \textit{entry} and follows the path to R3 (E-F-J-K-G). R1 then moves to location C, and from there it can detect the beacon at location B, so it moves a bit towards location B and drops a beacon set to \textit{explored} and then returns to location C, where it has one open path (location D). Another explorer is needed, so the process is repeated to bring R2 from location E past R1 to location D, R4 from location F past R2 to location H and R5 from location J past R4 to location L. Once there R5 detects R6 at location K. The current state of the dispersion is shown in Figure 3.6.

R5 then drops a beacon set to \textit{explored} and begins retracting. The robots retract along the path to the entrance (L-H-D-C-G-K-J-F-E-A), with each sentry waiting until the only robot it can hear is its sentry before dropping a beacon set
Figure 3.6: RDA staggered dispersion method example. The robots have fully explored the environment. Robots are circles with identifiers inside, unexplored beacons are squares and explored beacons are x's.

to explored and moving towards its sentry. This reverses the roll out process, but moves more quickly because multiple robots are moving simultaneously as they retract along the path to the entrance. The unexplored beacons at locations J, F, and E provide the path between R6 (at location K) and the entry beacon (at location A). R5 will stop at J, R4 will stop at F, and R2 will stop at E in order to maintain the chain of connected robots. When R6 passes R5, then R5 can set the existing beacon to explored and also retract. The same process follows for R4 when passed by R5, and R2 when passed by R4. R1 will stop at location A when it reaches it (the first robot to return to the entrance). From location A, the unexplored beacon at location B can be detected, so R3 (the second robot to return) approaches it (shown in Figure 3.7). R3 can then detect the explored beacons near location C and in doorway 2, so it sets the beacon at location B to explored and retracts to location A. The exploration is now complete, so when all
the robots have returned (R4 and R2 are likely still on their way back) the beacon at location $A$ is set to explored and the robots exit the environment.

Figure 3.7: RDA staggered dispersion method example. Retraction to location $A$ is complete, and R2 is investigating $B$. Robots are circles with identifiers inside, unexplored beacons are squares and explored beacons are x’s.

Chained Dispersion Method

After the first simulations, we realized that while the staggered dispersion method was preventing interference between robots when they were crowded into a small area at the start of the exploration, it was not allowing them to fully disperse into the environment the way we envisioned, and was a very slow stage of the exploration. It worked fine for a simple environment consisting of a corridor with small offices on each side, but proved problematic in complex environments, which are the target scenarios. We considered alternatives, and modified the dispersion method to allow for broader dispersion coverage of the environment,
while still trying to keep the robots from running into each other or getting grid-locked. In this method, the robots are distributed evenly between each open path leading out from the entry location. Each group disperses along their path to the edge of their combined communication range before needing to stop and request additional robots, or until they reach a dead-end and retract back to the intersection. Interference is reduced by having the robots advance as though they are connected by a chain and only following the robot in front when the chain pulls taut. The following example highlights the differences in how the robot team completes the exploration with the new dispersion method.

Once again, the robots all begin at location A, and R1 stays there as a sentry. That leaves 5 robots that can move in the environment, so R2, R5, and R6 move towards E, and R3 and R4 move towards B. R2 will stay at E as a sentry, while R5 and R6 explore to F. At the same time, R3 will stay as sentry at B while R4 explores towards F. R4 will detect R5 and R6, and leave an explored beacon between F and B as it retracts to B. R4 will then explore to C, while R5 acts as sentry at F and R6 explores to J. At this point, all robots are at their maximal dispersion in the environment, but there is still more to explore (Figure 3.8).

At this point, both R4 and R6 have a branch_count of 2, but the path to R6 is longer, so its path will be completed first, and the robots on R4’s path will be pulled back as a result. R4 will leave an unexplored beacon at C and follow the path C-B-A-E-F-J to the frontier. It will then explore to I, discover the dead-end and mark it with an explored beacon before retracting to J. R4 then explores to K and calls for additional robots. R3 drops an unexplored beacon at B and follows the path out to K. It then explores to L and calls for more robots. R1 responds, and makes its way to H, and then R2 follows to D. R5 follows the path from F to D, but detects the potential loop when it continues towards C, so
Figure 3.8: RDA chained dispersion method example. Maximum dispersion achieved. Robots are circles with identifiers inside, \textit{unexplored} beacons are squares and \textit{explored} beacons are x’s.

Figure 3.9: RDA chained dispersion method example. Exploration almost complete, a loop and a dead-end have been identified. Robots are circles with identifiers inside, \textit{unexplored} beacons are squares and \textit{explored} beacons are x’s.
drops an \emph{explored} beacon between \( C \) and \( D \), then retracts to \( D \) and then to \( H \). This state of the exploration is shown in Figure 3.9.

R5 now continues retracting to \( K \), where R4 still has an open path, so R5 explores to \( G \). R2 is in a dead-end, so after R5 has passed it completely, it drops an \emph{explored} beacon and retracts as well. When R2 has fully passed R1 at \( H \), R1 will drop a beacon and retract, and when it has passed R3 at \( L \), R3 will do the same. R2 follows R5 up to \( G \), then continues on to detect the \emph{unexplored} beacon at \( C \), so it leaves an \emph{explored} beacon between \( C \) and \( G \). R2 then retracts, joining R1 and R3. R1 will fill in at \( F \), R2 will fill in at \( E \), and R3 will fill in as sentry at \( A \). R5 will retract and then move out to the \emph{unexplored} beacon at \( B \). R6 will retract to \( A \) and wait (no additional calls will come in), as will R1. Meanwhile, R4 will have retracted to \( A \) and then will be moving out to the \emph{unexplored} beacon left at \( C \). This is the state shown in Figure 3.10.

![Figure 3.10: RDA shained dispersion method example. Retraction from first path exploration is almost complete, and R5 and R4 have moved out to finish the exploration of the \emph{unexplored} beacons. Robots are circles with identifiers inside, \emph{unexplored} beacons are squares and \emph{explored} beacons are x’s.](image)
After that R4 detects it is in a dead-end, and will retract to A. R2 also retracts to A, and once R4 has fully passed by R5, it will also retract to A. At that point, the exploration and retraction are complete. This method results in a very different path through the environment than the staggered dispersion method, but this method leads to faster and more coverage during the dispersion stage, as well as a faster overall exploration.

### 3.3 Sweep Exploration Algorithm

In the Rolling Dispersion Algorithm (RDA) the robots are either *explorers*, which move into the frontier, or *sentries*, which maintain a return path to the entrance. Each robot uses connectivity with its neighbors and distance to nearby obstacles to choose which behavior it will execute on each iteration of the algorithm. The messages between neighboring robots contains several potentially large pieces of information. and there are usually multiple robots moving at a time, exploring different parts of the frontier.

In contrast, the Sweep Exploration Algorithm (SEA), while originally based on RDA, is intended for use in scenarios with much more restrictive communication, such as chemical signals, or line-of-sight using a camera and color LEDs. This algorithm was developed in collaboration with Dr. Ken Sugawara at Tohoku Gakuin University in Sendai, Japan, as part of an NSF-funded program. It is a combination of RDA and Dr. Sugawara’s work on chemical-based communication to mimic ant pheromones. Using line-of-sight or chemical signals to communicate is a severe restriction, making it critical to reduce the number and size of messages in order to ensure full exploration.
In SEA, the robots decide their next action based on the states of their neighbors. There are two states used only by robots, one state used only by beacons, and five states used by both. The states are also the entirety of the message contents that are sent, meaning that we can represent each state with a binary encoding of only 3 bits (or colors, or chemicals), or as a full byte in which each bit corresponds to a different state. The eight states are:

**Branch:** Robot or beacon marking an intersection.

**Call Path:** Robot or beacon marking the path to the unexplored frontier.

**Explorer:** Robot moving along a path to answer a call, and then continuing the exploration of the frontier; or a robot answering a failure notification and repairing the gap in the network.

**Failure Path:** Robot or beacon marking the path leading to a location where an agent failed and needs to be replaced.

**Repel:** Beacon marking an area as explored, preventing repeated exploration.

**Retract Path:** Robot or beacon marking the path for a retracting robot to follow back to the previous branch.

**Retractor:** Robot has reached a dead-end, dropped a beacon to mark the explored area and is in the process of retracting to the nearest branch.

**Sentry:** Robot or beacon marking a path where there are no active notifications (call, failure, or retract).

However, this also means that only one robot can be moving at a time, or the messages get mixed up and parts of the environment may be missed. Therefore,
instead of the robots initially dispersing in any direction, as in RDA, they travel
one at a time down a single path, until it is completely explored, and then retract
and explore a new path. We show the finite state machine in Figure 3.11 and
provide the list of transitions in Table 3.1. The Repel state is limited to beacons,
and so is not included in the state machine or transition table.

A robot in the Explorer state will move away from the rest of the robots
into the unexplored frontier until it reaches a dead-end (see Definition 1). A
robot in the Retractor state will return along the path of robots and beacons to
the previous intersection (see Definition 2) and then become an Explorer again.
A robot in any of the other states will remain in place, marking the path and
providing other robots with the status of those robots further down the path
(passing along calls for additional Explorers, or notices of retraction or failures).
Robots in intersections use the state Branch to alert the Retractors that they
have reached the point where there is again a frontier to explore.

The robots communicate and navigate towards or away from each other by once
again making use of the communication signal strength. An Explorer following
a call path will move up the signal gradient towards the next Call-Path robot.
When the Explorer reaches the waiting robot, it will then move away from it
(continuing in the same direction it has been moving, unless at an intersection)
going down the gradient of the signal strength. When the Call-Path robot detects
that the Explorer is moving away, it will change its state to Sentry. The Explorer
then seeks out the next Call-Path robot to move up that signal gradient. If no
Call-Path robot exists, the Explorer will explore the frontier at the edge of the
communication range with the previous Sentry, until it detects either a dead-
end and initiates the retraction process or it determines that another Explorer is
needed and initiates the call process.
Table 3.1: Sweep Exploration Algorithm Transitions

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next State</th>
<th>Side Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explorer (E)</td>
<td>Intersection</td>
<td>Branch</td>
<td>Send call</td>
</tr>
<tr>
<td></td>
<td>Edge of Range</td>
<td>Sentry</td>
<td>Send call</td>
</tr>
<tr>
<td></td>
<td>Dead-end</td>
<td>Retractor</td>
<td>Drop Repel beacon</td>
</tr>
<tr>
<td>Sentry (S)</td>
<td>Retract</td>
<td>Retract Path</td>
<td>Send retract</td>
</tr>
<tr>
<td></td>
<td>Call—no child</td>
<td>Explorer</td>
<td>Drop Sentry beacon</td>
</tr>
<tr>
<td></td>
<td>Call—has child</td>
<td>Call Path</td>
<td>Send call</td>
</tr>
<tr>
<td></td>
<td>Failure—no child</td>
<td>Explorer</td>
<td>Drop Sentry beacon</td>
</tr>
<tr>
<td></td>
<td>Failure—has child</td>
<td>Failure Path</td>
<td>Send failure</td>
</tr>
<tr>
<td>Branch (B)</td>
<td>Dead-end</td>
<td>Retractor</td>
<td>Drop Repel beacon</td>
</tr>
<tr>
<td>Call Path (CP)</td>
<td>Explorer passed</td>
<td>Sentry</td>
<td></td>
</tr>
<tr>
<td>Failure Path (FP)</td>
<td>Explorer passed</td>
<td>Sentry</td>
<td></td>
</tr>
<tr>
<td>Retract Path (RP)</td>
<td>Dead-end</td>
<td>Retractor</td>
<td>Drop Repel beacon</td>
</tr>
<tr>
<td>Retractor (R)</td>
<td>At branch</td>
<td>Explorer</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.11: Finite state machine for robots using the Sweep Exploration Algorithm.
When the robots retract along a path, they follow the same process, with the Retractor robot leaving a beacon in the Repel state, and then moving up and down gradients as it passes the robots on the path, and those robots switch from Retract-Path to Sentry when the Retractor has passed them. When the Retractor reaches a Branch with additional paths to explore, it will become an Explorer and move down a new path. When it hits the communication limit, it will send a call. If a Sentry gets this call and detects that it is also in a dead-end, it will send back a retract signal, and retract to the Branch, at which point it will switch to Explorer and will follow the call path. If there are no Branch robots (everything has been explored), then the Retractor will retract to the location closest to the entry that is still in contact with the other robots, and will become a Sentry. The robot will then send a call to bring the other robots back to the entrance.

A Sentry robot in a dead-end will follow the same process as above, retracting closer to the entrance and initiating another call. This will repeat until all robots have retracted to the entrance, at which point the entry location will be detected as a dead-end, and the exploration and return are complete.

### 3.3.1 Algorithm Example

We again provide an example of the algorithm’s progression through the environment in Figure 3.4. The robots still all begin clustered at location A. R1 will be the Sentry at location A. R2 moves to E, then sends a call. R3 follows the call path to E, explores to F, and then sends a call. R4 follows the call path from A to E to F, explores to J, and sends a call. Following the same process, R5 follows the path out to I, where it will detect that it is in a dead-end. It then initiates a retraction, dropping an explored beacon and returning to J. From there, R5 explores out to K, and sends a call for another Explorer. R6 responds to the call.
and follows the path \(A-E-F-J-K\), then explores to \(L\) and sends a call. All of the robots are now in play, as shown in Figure 3.12, but there is still a large amount of the environment to explore.

Figure 3.12: SEA progression example. All robots are out in the environment, and one dead-end has been marked. Robots of any state are circles with identifiers inside, \(Sentry\) beacons are squares and \(Repel\) beacons are x’s.

When R6 sends a call, R1 drops a beacon set to \(Sentry\), and follows the path \(A-E-F-J-K-L\) and then explores to \(H\) and sends a call. R2 follows the call and then explores to \(D\); R3 then makes its way to \(C\); and R4 follows and goes to \(B\). R5 follows the path first from \(K\) to \(B\), detects the \(Sentry\) at \(A\), so drops a \(Repel\) beacon between \(A\) and \(B\), and retracts to \(B\). It repeats the process between \(B\) and \(F\), again leaving a \(Repel\) beacon to block the loop. \(B\) is now a dead-end, so R5 retracts back to \(C\), then explores to \(G\), and makes a call. This is the state shown in Figure 3.13.

R4, being in a dead-end, leaves a \(Repel\) beacon and responds. R4 discovers the loop that would form between \(G\) and \(K\), leaves a \(Repel\) beacon to block it,
Figure 3.13: SEA progression example. Exploration is almost complete, and two loops have been blocked (the third loop has not yet been detected). Robots of any state are circles with identifiers inside, *Sentry* beacons are squares and *Repel* beacons are x’s.

Figure 3.14: SEA progression example. Exploration is complete, and the robots are in the process of retracting to the entrance. Robots of any state are circles with identifiers inside, *Sentry* beacons are squares and *Repel* beacons are x’s.
and retracts along the path to $K$. When it reaches $K$, it sends a call. $R5$ is in a dead-end, so it leaves a $Repel$ beacon and retracts to $J$, then sends a call. $R3$ is in a dead-end, so it leaves a $Repel$ beacon and retracts to $F$ before sending a call. This state is shown in Figure 3.14. This process of leaving a $Repel$ beacon is repeated until all of the robots retract to the entrance, which brings them back in the order $R1, R6, R4, R5, R3$, and $R2$. At this point, there are $Repel$ beacons at each of the letter locations, as well as between $A$ and $B$, $B$ and $F$, and $G$ and $K$. 
Chapter 4

Algorithm Correctness and Properties

This chapter covers the formal proofs of correctness, showing that the algorithms will complete the exploration under our restrictions. We also provide discussion of various other properties of the individual algorithms, including the minimum number of message types for SEA.

4.1 Correctness

We present here formal analysis that the robots running our algorithms will, even with communication restrictions, complete the exploration without missing any point in the environment, will not end up in infinite loops (so that the exploration will be completed and the robots will exit when done), and can succeed in these goals even with robot and beacon failures.

Though SEA has more restrictions on communication than RDA, both algorithms operate in the same manner at their core, so the following properties and
proofs apply to both algorithms. The differences in the algorithms show in how the robots move (multiple or one at a time), and how much information is shared between robots.

**Lemma 1.** *The algorithms avoid unnecessarily repeated exploration.*

*Proof.* We will do this proof in two parts: first assuming that the beacons do not fail and then assuming that they do fail. In either case we will prove by contradiction that the algorithms will avoid unnecessary repeated exploration.

First, assume that the robots explore an area that has been previously explored. This produces an immediate contradiction because, when an exploring robot reaches a dead-end it drops a beacon to mark the area as explored. Any re-exploration is prevented by the presence of this beacon.

Second, in the case when a beacon marking an explored area fails, the area around that beacon becomes unmarked. If the failed beacon is surrounded by beacons marking the path as explored, then no robot will reach that area, so it will not be re-explored. If, however, the beacon was bordering unexplored regions, then the robots will have to re-explore the now unmarked area until reaching a dead-end, and then once again mark the area as explored, preventing future unnecessary visits. The important caveat here is that we need to assume a finite number of beacon failures, otherwise robots would re-mark areas infinitely, which would lead to other areas not being explored or the robots not returning to the entrance to report the completed exploration. Therefore, given a finite number of beacon failures, which is a reasonable assumption, we again derive a contradiction.

**Lemma 2.** *The algorithms avoid infinite loops.*

*Proof.* We will again consider two cases in this proof: first assuming that the
beacons do not fail; and then assuming that they do fail. In both cases, we will prove by contradiction that the algorithms will not get stuck in infinite loops.

First, assume that a robot is repeatedly exploring a loop in the environment. This is immediately a contradiction of Lemma 1 because the point at which the exploring robot first closed the loop (by reaching a previously explored location), it would have detected it was in a dead-end, retracted a small amount, and then dropped a beacon to mark the area as explored. That beacon will block the loop, since the robots will treat it as an impassable obstacle no matter the direction from which they approach.

Second, in the case when a beacon fails, the area surrounding the beacon’s location becomes unmarked. If there are still surrounding beacons marking the area as explored, then there will be no effect on the robots’ exploration. If the beacon is neighbors with a robot or the path to the frontier or entrance, then the area will be re-explored, as in Lemma 1, but will again be marked as explored when a dead-end is once again located. Therefore, given a finite number of beacon failures, which is a reasonable assumption, we again derive a contradiction. \(\square\)

**Lemma 3.** The algorithms achieve full coverage with a single robot.

**Proof.** Assume we have one robot and an infinite number of beacons. In both algorithms, the robot will advance, leaving beacons to mark the return path and areas that have been explored. The robot will explore each path until it reaches a dead-end, and then it will retract to the nearest previous branch that still has unexplored paths, and continue to explore/retract behaviors until it has explored the full environment and retracted to the entrance. For a single robot, the explore/retract process is equivalent to Depth-First-Search (DFS), which is proven to be complete in a finite search space when repeated states and loops are avoided [86]. By Lemmas 1 and 2, we have proven that our algorithms avoid
repeated states and loops. Our environment is finite. Thus, our algorithms will achieve full coverage with one robot.

\[\text{Theorem 1.} \quad \text{The algorithms will achieve full coverage of the environment with multiple robots, and all functional robots will return to the entrance.}\]

\[\text{Proof.} \quad \text{We prove by induction that the algorithms function correctly when multiple robots are used.}\]

\[\text{Base Case:} \quad \text{The base case is that the algorithms achieve full exploration when only one robot explores the environment. The proof is given in Lemma 3.}\]

\[\text{Induction Step:} \quad \text{Assuming that the algorithms achieve full exploration with} \ k \ \text{robots and the remaining functional robots return to base, we want to prove that the algorithms achieve full exploration when one more robot is added. Suppose that there are} \ k+1 \ \text{robots, then we need to show that: (1) the robots will not continuously explore overlapping areas, and (2) that the robots will not miss an area because they lost contact with the other agents, and (3) that no robot will be stranded.}\]

\[\text{First, in Lemma 1, we have already shown that there will not be unnecessary repeated exploration of an area, so long as the beacons remain active. Every robot that approaches the area will detect the beacons and move to a different area, and this remains true no matter how many robots are added.}\]

\[\text{Second, both algorithms enforce the restriction that the robots remain in contact with at least one other agent at all times, and when that connection is lost, the robots will immediately stop exploring and retreat in order to reconnect. This is required for ensuring complete coverage, because the connectivity means that robots will not miss any area, and will not re-explore areas previously covered. This also remains true no matter how many robots are added.}\]
Third, the connectivity keeps the robots from being stranded in the environment, because only the exploring robot in a dead-end will mark an area as explored. The retraction step then brings the robot back into the group before the next robot marks an area as explored, so that no robot is left behind or trapped. In addition, the fact that the robots explore a path and then retract to the previous intersection, and then repeat the process, similar to Depth-First-Search (DFS), means that all the still functional robots will eventually retract back to the entrance when the exploration is complete. Once again, adding another robot to the team does not change this functionality.

Thus, with \( k+1 \) robots, the algorithms achieve full exploration, and the remaining functional robots return to the entrance when the exploration is complete.

4.2 Algorithm Properties

In addition to the previous proofs, there are several important properties of the algorithms. Using multiple robots reduces the individual load on each robot, but the coordination adds costs in location visits and amount of information passed in messages (as well as number of messages sent). We discuss here the upper bounds on those costs.

4.2.1 Repeated Visit Limits

We can represent the environment as a graph, in which nodes represent locations that are separated by the distance of the communication range, and edges exist along the paths that robots can travel between vertices. This limits our graph to a finite number of nodes. We show a possible graph as an overlay on the simulation.
Figure 4.1: (left) Cave-like environment used in simulations, showing overlay graph. (right) One possible exploration using four robots. The star marks the start, and filled shapes are where dead-ends are detected on a path, initiating the retraction process.

The image on the right shows one possible path when four robots are exploring the environment using RDA. One robot will stay at the star while the other three move along the three open paths from that intersection. All three will reach the edge of the communication range and send a request for additional explorers. The robot on the left edge has the lowest \textit{branch\_count} and the robot at the square has the most, so the robot at the square will drop an \textit{unexplored} beacon and go back to the star and then to the left edge. The dead-end in the upper left corner will be discovered, and the exploration will continue on the path leading down and around the bottom of the environment until it reaches the filled circle, from which the beacon at the square can be detected, making that filled circle a dead-end on
this exploration path. The upper right path will be explored next, and finally a robot will return to the square and detect that it is now surrounded by explored beacons, making it a dead-end and the last location to be marked as explored.

Returning to the left image of Figure 5.22, assume we have only a single robot. During the exploration, edges that would lead to loops will be removed by marking them with explored beacons (in RDA) or Repel beacons (in SEA). With a single robot, each node is visited at most once per edge, assuming there are no failures. Placing the beacons to block loops causes an additional visit to the previous node, but not the node on the other side of the beacon. Dead-ends are equivalent to leaf nodes, and are visited only once, and each intermediate node is visited when the robot first gets to that point, and again each time the robot returns from a path. With multiple robots, each intermediate node $n$ is visited at most $e_n \times n'$ times, where $e_n$ is the number of edges node $n$ has, and $n'$ is the number of nodes beyond node $n$, summed over all paths that branch off of that node. If there are fewer than $n'$ robots, then we can also say that $n'$ is bounded by the number of robots.

4.2.2 Minimum Message Types

In RDA, the robots must send many messages to confirm that they are still within communication range of each other, since multiple robots can move at the same time. This is quite costly in terms of bandwidth, processing, and power consumption, and may not be feasible with some kinds of communication. If we use chemical signals, flooding the environment with those chemicals will cause us to lose new messages in the old ones. But in restricting the communication in SEA, we lose the ability for multiple robots to move at the same time. However, it
does put bounds on the number of messages being sent, which makes the coordination easier as well. In SEA, we require only eight message types to complete the exploration, which correspond directly to the eight states listed previously. Because of this, the robots can complete the exploration with very little required in terms of information sharing, which is often one of the more expensive aspects of multi-robot systems.

Consider starting with only two messages/states. The explorer and retract path are the most straightforward, and both are needed in order to create the backbone of paths to the frontier and back to the entrance. But there is no way to request additional robots to the frontier when the last explorer reaches the edge of the communication range with its nearest neighbor. So we need a third state, the call path, which both makes the request and leads the new explorer to the frontier. But then the explorer hits a dead-end, and without the repel state, the robots might explore the same dead-end repeatedly, or go into an infinite loop around an obstacle. This brings us to four states. As a robot retracts along the retract path, it needs a way to differentiate between the robots on the path that it has passed, and the robots it still needs to pass. So the robots that it has passed need to switch to the sentry state, otherwise the robot might oscillate between two robots on the retract path and never make it anywhere else. Then, without the retractor state, the agents on the retract path wouldn’t know when to either change to the repel state (for beacons) or to switch to the sentry state and prepare to retract themselves (for robots). Without the branch state, each retractor robot would go all the way back to the beginning, and a branch might be cut off from further exploration because the retraction protocol requires that the robots leave repel beacons along the way. Since we are working in an unknown and potentially dangerous environment, it is essential that we have a means of identifying and
replacing failed agents along the frontier and path back to the entrance, making
the fail path an critical state. Thus, we require eight messages to complete the
exploration.

Given this evidence, eight message types are required to complete the explo-
ration under the conditions of our specified problem. If we were to accept that
there would be no failures (an implausible assumption in a disaster scenario), we
could remove the fail path. If we required the robots to always turn the same
direction (if possible) at an intersection, it is possible we could do without the
branch state, but then the retract path would be sent all the way to the entrance
for every retraction that occurs, as would the call path messages. This is currently
avoided by having the branch message stored in the memory of each neighboring
agent, and in combination with either a call or a retract, the agents can correctly
direct the path. Removing any one of the message types restricts the algorithm
and its ability to achieve complete coverage in a way we feel is important for
disaster scenarios.

Additionally, using only eight message types is still highly restricted. While
we are not currently considering adversarial environments, this algorithm could
be used in such situations, such as for reconnaissance in a hostage situation or
something similar. In that case, the restrictions on the number of message types
makes it difficult for opponents to break into or corrupt the system. If the robots
are sharing at most a single byte of information at a time, there isn’t much for
the opponent to decode, and the majority of the functioning is kept internal to
each robot. This provides reasonable protection against interference in many of
the situations we have envisioned for the algorithms. In an adversarial setting,
this encoding would be broken with a bit of effort, but even then the minimalism
will buy some time for the robots to explore unimpeded.
4.2.3 Handling Failures

There are many ways in which robots or beacons can fail during the exploration of an unknown environment, particularly one that is considered dangerous, such as the aftermath of a disaster. This includes damage to the agents, causing the agent to fail completely (cease all actions) or in part, such as a malfunctioning motor, sensor, or cessation of communication. The last is particularly concerning for our algorithms, since they rely on communication to complete the exploration. Where the agent is located in the environment, and particularly its role in the current exploration also has an impact on the overall performance of the exploration. We here cover how the algorithms handle each type of failure.

First, we will deal with partial failure of an agent. For robots, this takes one of three forms (or combination thereof): movement, perception, or communication. There are many ways in which a robot can cease to move as it should. We do not mean simple odometry errors, which are frequent and most robots can adapt to, but possibilities such as a motor no longer turning, wheels on one side falling into a hole and being unable to get back out, or some other large failure. In each of these cases, the robot can detect the malfunction, because it will not see any change to the communication signal intensity when it attempts to move away from another agent. Both algorithms have a time-out for this type of situation. When one of these movement failures is detected, the robot becomes a sentry in the environment, with the appropriate status to bring new explorers to complete the exploration of the area. When the exploration has progressed to the point that the robot is on the edge of the robot group (so it has robots on one side, and beacons on the other), it begins reporting that it is a beacon, because it will not be able to move to other parts of the environment and so needs to not be considered in the remaining progression of the exploration. In the event that multiple robots
in a row (up to 3 or \( n/3 \) where \( n \) is the number of robots, whichever is smaller) enter the area and then become stuck or damaged and switch to being sentries, the area will be considered inaccessible to the robots and will be treated as a dead-end. Any additional exploration in that area will have to wait for either a different type of robot, or the human rescue workers.

For robot perception failures, the robot may not be able to internally detect the functional state of its sensors. However, when near other robots, it can compare sensor data and determine that its sensors are not responding as they should. This is particularly true for proximity sensors. In the event of perception failure, the robot will become a sentry at its current location, and, when the exploration puts it at the edge of the robot group, it will act as a beacon. While it might still be possible for the robot to follow the path to the entrance using only the communication signal, it is better for it to remain in place so that it doesn’t get stuck in a way that would block the other robots from continuing. Alternatively, but not yet implemented, it might be possible in RDA for the robot to attempt to return once the other robots have all passed it, following them to each branching point and only continuing when the next beacon between it and the entrance becomes an \textit{explored} beacon (signaling that there is nothing more to explore beyond that beacon). We do not currently see a way to implement this type of functionality for SEA. Should multiple robots in a row (up to 3 or \( n/3 \) where \( n \) is the number of robots, whichever is smaller) fail in the same general area, that area will be treated as a dead-end and avoided. Losing up to 3 robots in the same area implies that the area is inaccessible to the robots, and we do not want to commit additional resources to be lost.

Communication failures between explorer robots (those moving on the frontier of the environment) and their neighbors (stationary and moving) can come
in three main types. First, there is the possibility that a robot has moved beyond the communication threshold. This can happen when there are varying levels of interference in an environment; so a robot can have good signal intensity and then move only a few inches and completely lose communication. Second, there could be a change in the environment (ceiling or floor collapse, for example) that interferes with a robot’s ability to communicate. Third, a robot’s communication method itself could fail. This will effectively prevent the robot from being able to continue the exploration, because it won’t be able to communicate with either other robots or beacons. However, if the robot cannot detect that it is an internal fault, it has no way of telling the internal failure from the other types of failures. In all three cases, the algorithms have protocols for handling a loss of communication. First, the exploring robot that lost communication will reverse direction for a short distance (maximum of one meter). If this fails to reestablish communication, it will turn around and explore back in the direction it came from, until it either reestablishes communication, suggesting it was the first type of failure, finds a dead-end, suggesting it is the second type of failure, or a time threshold is exceeded, suggesting the third type of failure. The time threshold is 3 times the amount of time needed to traverse the maximum communication distance threshold with no communication interference. In many of the RDA experiments, this is set to 10 meters, and the robots know the time it takes them to traverse that distance (in a straight line). If the robot is able to reestablish communication, it will stop at that location and request additional explorers to continue beyond it. If the robot finds a dead-end, it will start from that point and begin exploring as though it is the only robot, but leaving behind beacons to block the area it came from, since there is no entrance there. When it finds a branch, it will explore each path leaving unexplored beacons to lead it back to the branch (switching
the beacons to explored as it retracts to the branch), so it can continue exploring the remaining paths. If it can reconnect with the team at another place in the environment (where a loop would have been detected before the blockage), it will rejoin the team, and any path it completely explored from the blockage to that connection will be marked as explored already. Those that had not been explored will be explored when the team gets back to them. In that case, the robot will be able to retract with the team to the entrance. If it cannot reconnect, it will stop when it blocks itself into a dead-end. This still leaves important information and a network of beacons in the environment for the human search and rescue team, when they can get into that area, but that information cannot be passed along until they get there. From the other agents’ perspective, when the explorer robot stops communicating with them, they will wait for the time threshold to be exceeded, and if the explorer has not gotten back in communication by then, the other agents will call for another explorer to replace the missing one. Once again, should there be multiple failures in a row (up to 3 or \( n/3 \) where \( n \) is the number of robots, whichever is smaller) in the same area, it will be treated as impassable and marked as a dead-end.

For the purposes of our algorithms, these protocols to reestablish communication or to bring in other robots to replace a failed or missing robot are sufficient to complete the exploration. Should a robot fail completely, by falling down an open elevator shaft or having a ceiling collapse on it, for example, the algorithms respond in the same way as for a loss of communication. The neighboring sentry waits for the time threshold, then requests a new explorer. After multiple failures in a row (up to 3 or \( n/3 \) where \( n \) is the number of robots, whichever is smaller), the area is considered impassable and marked as a dead-end. This still allows us to guarantee full coverage of the accessible environment, given the robots’ movement
Beacon failures involve a slightly different set of scenarios. Robot failures happen while the robots are still connected to the rest of the group, and so replacing them with other members of the group is relatively straightforward by changing the failed robot’s status (if possible) or giving them up as lost. Any beacon on the edge of the robot group or beside an unexplored path when it fails will be detected by the robots, and the robots can either replace it immediately (for beacons next to robots), or when they pass that area again (for beacons on or beside an unexplored path). Beacons, however, may also fail in areas where the exploration is complete, or in the middle of the path to the entrance. In either of those cases, there are no robots nearby to bridge the gaps. For the failed beacon on the path to the entrance, the robots need to fill that gap with the knowledge that it is their return path. For SEA, this is accomplished by the robots following the fail path, which means they do not mark the area they retract from as explored, and they place a new Sentry beacon in place of the failed beacon, repeating as many times as necessary to bridge the gap in the path (when they find the fail path beacon on the other side, which detects the robot explorer and reverts to its previous state of Sentry). A similar process is followed for a failed beacon in the middle of an explored area, but when the robots reach the other side of the gap, the beacon there will revert to its previous state of Repel, and the robots will then retract, changing the beacons to Repel until they reach the nearest branch. For both cases, the robots then move back out to the frontier to continue the exploration. RDA will wait until it is returning from exploration beyond the failure or branching point to the failure to bridge the gap. This is possible because it has the exact path to the failure point, and can share path information with beacons that had been neighbors with the failed beacon to rebuild the path to the entrance during.
the return.

While we could ignore the beacon failures when they happen in an explored area, with other beacons still marking either side of the failed beacon as explored, we prefer to replace the beacons to bridge the gap in order to maintain the complete network, which is passed to the human search and rescue team for their planning and subsequent movement through the environment. Between these reconnection and replacement actions, we are able to maintain the communication for the duration of the exploration. The only situation that the algorithms cannot accommodate is if the environment changes in a way that completely blocks the robots off from the entrance. In that case, the robots will still complete their exploration, but unless they are able to detect another entrance, they will not be able to share their information without the human team finding a way to establish contact with the stranded robot team. It is possible that a loop may have been blocked off, but if reopened it would provide an alternate route to the entrance.

We have not implemented this as yet, but it would be simple to add to RDA, which can track and exchange information about loops. It would not be possible in SEA, because the loops are not recognized as such in SEA, they are merely seen as dead-ends, so there would need to be major alterations to make SEA capable of opening a closed loop, without doing a reset of all beacons and exploring the entire environment again.

An impassable blockade that cuts the robots off from the entrance or the case where all robots are destroyed are both situations that few, if any, algorithms can recover from. Barring either of these extreme failures, our algorithms are able to handle individual or small groups of agent failures and complete the exploration of the accessible areas of the environment. Thus, our guarantee of full coverage holds under the conditions specified.
Chapter 5

Simulations and Experiments

We conducted our simulations in the Player/Stage [46] environment. Each simulation used the Pioneer robot model for the mobile robots, and a modified Pioneer robot for the beacons. Each robot was equipped with 16 sonar sensors and a laser range-finder for obstacle detection. We ran simulations in multiple environments with varying layouts. The robots start from the same location for each run in each environment. The robots choose their directions with some randomness to avoid hitting each other, especially at the start of each run, so results are averaged over at least 15 runs of each combination of environment, algorithm, and number of robots.

For the physical robot experiments, we initially used the Scribbler robot with the Fluke extension [52], and later we used the LEGO EV3 robot [63]. The EV3 has a USB wireless dongle attached, as well as a micro SD card, which contains ev3dev [36], a distribution of Debian that has been pared down to run on the EV3. Using ev3dev, we are able to access the full scope of the robot’s built-in functionality, but also write more advanced programs using Python.
5.1 Preliminary Simulations

5.1.1 Simple Office Environment

The first set of experiments used the environment in Figure 5.1 to test the main properties of the algorithms. The environment has a simple topological structure, but it is large enough to require nine robots for blanket coverage. In this environment, with its long corridor and small rooms off each side, both methods performed identically with a single robot, and nearly the same with two robots, because the two robots are constantly maneuvering around each other to continue exploring. This maneuvering is the reason the multiple robot experiments take longer to complete, and travel further than the single robot experiment. However, with four robots (shown in Figure 5.2) RDA took significantly less time, even though the robots traveled approximately the same total distance as the four robots running SEA. While the robots using RDA could explore in multiple directions simultaneously, the robots using SEA had to wait for each individual path and room to be cleared, and then had to move back to the corridor one at a time. Results are the average over 15 runs for each combination.

5.1.2 Cave Environment

In the second set of experiments, we used a cave-like environment, which is more complicated in part because of how open the area is, and how many cycles are possible in the exploration. Though the area to be covered is smaller than that of the simple corridor environment, the cave environment requires ten robots (one more than the simple environment) for blanket coverage due to the many corners and odd angles of the obstacles. We ran these experiments 10 times per algorithm with five and eight robots.
Figure 5.1: The Simple Office environment with exploration partially completed. The four robots are centrally located, and are moving right to left. Two beacons have been activated in the doorways of the rooms on the right.

Figure 5.2: Total distance, average distance per robot, and total time for simulations using one, two, and four robots with each algorithm in the Simple Office environment. Results are averaged over 15 runs for each combination.
Figure 5.3: Average time to full exploration over 10 runs for simulations using one, five and eight robots with each algorithm in the cave environment simulation.

Figure 5.3 shows the rate of coverage for the RDA and SEA algorithms in the cave environment. While in most cases the simulations show a fairly steady increase in the percentage of coverage, the RDA with five robots does show a plateau, due to the fact that the robots initially disperse in all directions, and then must wait for others to leap-frog out to the frontier. The SEA algorithm does not have this plateau because the robots explore along only a single path at a time. The experiments using SEA start with a higher initial coverage, due to the fact that the algorithms use different initial dispersion methods, and using the exact same cluster at the start led to many robot collisions. Accounting for that difference, and comparing times from the same starting percentage to full coverage, SEA is 1.35 times faster at achieving full coverage than RDA. SEA outperforms RDA in environments where there are long paths, because SEA fully
explores only one path at a time, while RDA will attempt to explore many paths simultaneously, but will run into delays when robots have to be pulled from other paths to fully explore long paths.

Figure 5.4 shows the simulation view and coverage map for the start and end of a simulation run with five robots. In the simulation view, one can see the five robots as well as their sensors’ field of view. The coverage maps show the area that had been viewed by the robots’ sensors, with locations that were viewed multiple times shaded darker than locations that were viewed only once over the course of the exploration.

5.2 Physical Robot Experiments

5.2.1 Scribbler Robots

In our first experiments using physical robots, we used the Scribbler robots with the IPRE Fluke [52] attachment. We made use of the camera to read beacons and IR sensors to locate intersections. The Scribblers are only able to wirelessly communicate over Bluetooth, and they are further limited to a single open channel, meaning all communication had to be routed through a centralized computer, which greatly reduced their efficacy for our algorithms. Due to the limitations of the robots’ communication medium, only SEA could be implemented, as coordinating multiple communication channels at a time led to messages being received in the wrong order or too spread out in time when using RDA. The Clearpath Jackal [28] would be an appropriate robot for actual search and rescue operations. We later used other robots to perform further experiments with all of our algorithms.

Our experimental environment has a well structured lay-out, similar to a house
Figure 5.4: Five robots running RDA in the cave environment.

or apartment floor-plan, with small doorways and multiple rooms and hallways, shown in Figure 5.5. We ran five experiments each for one, two and three robots, which was the maximum number of robots that could maneuver given the size of the environment. We measured the time to complete the exploration, number of messages sent, and total distance traveled (see Figure 5.6). SEA does not perform well in this environment due to the restrictions on movement. The short paths
mean that the robots spend a significant amount of time just maneuvering around each other as they leapfrog back and forth, and also leads to a large overlap in areas covered by each robot.

5.2.2 LEGO EV3 Robots

After discovering the incompatibility of the Scribbler robots with our algorithms, we researched a better option for experiments on physical robots, that was also available to us in sufficient quantities. Of the mobile ground robots available to us, the LEGO EV3 [63] provided us with the best combination of cost, size, number, and flexibility of communication and computation. This is primarily due to the fact that we are able to connect a USB wireless communication dongle, and access the base functionality of the robot using the Python programming language through the ev3dev operating system [36]. This allowed us to fully implement both RDA and SEA, and run experiments in a large section of the NHH Building’s 1st floor (see Figure 5.7). The area in green highlights the open passageways that the

![Diagram of experimental setup](image)

Figure 5.5: (left) Overlay graph of experimental set-up (dotted line denotes start area). The star is the first point in the environment, and the filled circles mark dead-ends. (right) Experimental set-up with cardboard walls. The robots are all located in the start area on the left side.
Figure 5.6: Experimental results averaged over five runs with one, two, and three robots using SEA.

robots can access. The long (top horizontal) hallway is approximately 40 yards long, and the connecting (vertical) hallway is about 20 yards long. The loop off of the connecting hallway can be closed at the top or the bottom by doors, providing us with 3 possible configurations for the robots to explore (completely closed off, one door open creates a dead-end, and both doors open creates a loop). The size of the environment also allowed us to test the robots’ ability to complete the exploration when they are not all connected (due to distance and interference from building walls).

For the purposes of these experiments, we designed the EV3 robot to use a bump sensor on the front, an ultrasonic sensor on a swivel mount (so that it could monitor its distance from a wall while moving, but also check the distance to obstacles around it when stopped), and a color sensor to read the beacons.
Figure 5.7: Experiment environment on the 1st floor of the NHH building.

Figure 5.8: (left) The LEGO EV3 robot used in experiments. (right) Three of the beacons used in the experiments.

Overall the EV3 robot fits into an 8” x 11” x 6” box. We did not have access to wi-fi enabled motes for these experiments, so instead we created 1” x 2.3” x 0.25” plastic pieces with colored stripes in binary ID patterns for differentiation. These fit in the black beacon distribution holder mounted on the back right flank of the EV3 robots. The robots power a motor to drop off a beacon, then drive past it
so the color sensor can read the beacon’s ID pattern and update the neighboring robots with the beacon’s ID. The robot and three of the beacons are shown in Figure 5.8. Because the beacons need to be able to be read from either direction, not all binary values can be used. To give us more options, we used different colors instead, which can be differentiated by the EV3’s color sensor.

We provide the results of these experiments in Figure 5.9. Total distance increases with additional robots, due to the multiple robots traversing the same area to reach new frontiers. However, we do see the trend of decreasing average distance, and thus more load balancing between robots, as in the simulations. RDA with three robots approaches the time of a single robot, but SEA performs no better with three robots than with two robots in terms of time.
5.3 Comparisons with other Algorithms

In our later experiments and simulations, we implemented two algorithms from other researchers for comparison with our algorithms [55]. These two algorithms differ from ours primarily in terms of the communication constraints and the method of exploration, but were still attempting to complete the same unknown environment exploration problem as we are. With slight modifications to our own communication policies, the two comparison algorithms were similar enough to provide reasonable comparison results with our Rolling Dispersion Algorithm. The comparison algorithms and our modifications are detailed below, followed by the experiment results. Due to the extreme communication limitations in the Sweep Exploration Algorithm, we have chosen not to use it in these comparisons, as it is always slower to complete the exploration task than the other algorithms in complex environments.

5.3.1 Multi-Robot Depth-First-Search (MR-DFS)

The algorithm most like RDA is the Multi-Robot Depth-First-Search (MR-DFS) presented by Brass et al. [19]. This work, which includes theoretical and simulation results, implements Depth-First-Search for multiple robot teams on undirected graphs, which are then re-arranged as trees. It is assumed that the distance along an edge between vertices is uniform and can be traversed in a single time-step. At any vertex, the robots divide among the edges such that the difference between the number of robots advancing along any two of the edges leading away from the vertex is no more than one. Brass et al. envision that the robots leave markers at each vertex to inform subsequent robots which direction the earlier robots have gone, so that the following robots can take different paths. This does assume that
the robots can determine and clearly pass on the information about what direction they have gone, and that the beacons are not moved or bumped, as that could invalidate the information they store (such as which paths have been explored, in relation to the orientation of the beacon when it was placed or passed by a robot). There is no direct communication between the robots, however, so there is also no need for communication restrictions.

In comparison to our approach, MR-DFS allows the robots to split up and completely lose contact with one another. This leads to parts of the environment being needlessly explored multiple times, or robots traveling down a path where help might be needed, only to arrive after all the remaining unexplored paths have been fully explored, which is somewhat inefficient. In contrast, RDA is slower to reach full coverage, because the robots must stay in communication with each other, but travel less distance overall, because they don’t go down a path unless it is necessary. Areas that are redundantly covered due to the need for multiple robots to traverse the start of long, multi-branching paths are redundantly covered in both algorithms.

Figure 5.10 comes from the original journal paper, and shows the paths of two robots exploring a graph environment at multiple time steps. A worst case scenario with four robots exploring a tree environment is shown in Figure 5.11, again at multiple time steps as the robots explore a tree. This figure particularly highlights the issue with duplicate coverage due to the robots only communicating when they meet at vertices. Both figures and their captions are from Brass et al. [19].
Figure 5.10: Path of two robots at (a) five steps, (b) eight steps, (c) 11 steps, and (d) 15 steps. [19]

5.3.2 BoB

Viet et al. [96] implemented a multi-robot algorithm to achieve online complete coverage using a combination of the boustrophedon motion plan, in which the robots move along straight paths and then double back right beside the original path, much like an ox plowing a field (from which the motion plan takes its name), and backtracking using Greedy A*. The original boustrophedon motion plan ensures that the robots cover everything in their area, and the backtracking allows them to quickly move to an open frontier when they reach an end point in their area (due to obstacles and completed paths of the other robots). It does rely on the robots being able to globally communicate their paths and maps of their areas as they explore, but the map is not needed in advance, making it a
Figure 5.11: Decreasing branching property: worst case scenario. Path of the robots after (a) one step, (b) three steps, (c) six steps, (d) eight steps, (e) ten steps, and (f) 12 steps. [19]

good comparison for our algorithm alongside MR-DFS. Once again, they assume global communication (they discuss using only local communication, but in the simulations, it appears that the algorithm was only tested with robots that could all communicate with each other at all times, to make the Greedy A* backtracking feasible and efficient).

Figure 5.12 shows 3 robots exploring an open environment with multiple obstacles. Each robot’s path is shown, along with starting and ending locations at that point in the exploration. Note that robot $A^1$ is in a dead-end and needs to calculate where to move to in order to reach a new frontier. Figure and caption
Figure 5.12: Boustrophedon paths created by robots $A^1$, $A^2$, and $A^3$. [96]

are from the original paper [96]. Because it is an online coverage algorithm, only the cells that have been explored are shown, with the remaining cells being drawn as the robots complete coverage of those cells.

5.3.3 RDA Modifications

MR-DFS allows the robots to move further apart than in RDA, because there is no direct communication over distance between robots. Thus, our first modification was to double the communication range of RDA, from 10 to 20 meters, creating RDA-EC (extended communication). An extended communication range does require a better communication device, and more power to run that device, but in this case we are still on the inexpensive end of the communication device range.
Allowing RDA-EC robots to move further apart than the original RDA range changes the exploration pattern and improves the speed of coverage, while also reducing the total distance traveled. This also starts both MR-DFS and RDA-EC on more equal footing, so the comparison between them is more meaningful.

We also found the multiple starting locations in BoB appealing, especially since RDA showed difficulty at the start when too many robots were fighting for space to move, and also ended up doing redundant coverage in the initial area. We created the extension, RDA-MS (multi-start), in which each starting location held a separate team, which then ran the regular RDA algorithm, in order to more accurately compare BoB to our algorithm. One interesting discovery that we made from this modification and the subsequent experiments is that teams executing RDA with no knowledge of each other are able to still communicate with each other and the other teams’ beacons when close enough to each other, and that the only effect this has is that the robots will not enter an area that another team has already explored. This leads to some load imbalances if the teams encounter each other in ways that quickly stop one team (a case in which the BoB algorithm has the advantage, because the individual robots can move to an entirely new frontier, while the RDA robots cannot), but also greatly reduces the time to fully cover the area while still providing the full coverage guarantee.

5.3.4 Comparison Metrics

To evaluate the algorithms, we reviewed the metrics used to measure coverage algorithm performance. The most common are listed in Table 5.1, which also indicates which of the algorithms used each metric (RDA and its modifications use the same metrics).
Yan et al. [100] produced an in-depth evaluation of metrics for multi-robot exploration. They also discussed the difficulties inherent in assessing and comparing multi-robot algorithms, due to the high number of variables involved in each system. These include the type, size, capability, and number of robots, as well as many facets of the environment. They list five recommended metrics.

**Exploration time** Time from start of exploration until a threshold is met.

**Exploration cost** Sum of distance traveled by each robot.

**Exploration efficiency** Ratio of area covered to distance traveled.

**Map completeness** How much of the environment is covered.

**Map quality** How accurate is the map generated.

Considering these metrics, we see how they make sense for the general multi-robot exploration case, but some of the metrics don’t matter in our specific scenario. For example, we aren’t generating a map, so the quality of the map doesn’t apply. Map completeness does apply, but is always 100% for the algorithms we are comparing (they all achieve complete coverage). Exploration time corresponds to the metric in our table of time to coverage. We also include time to return.

<table>
<thead>
<tr>
<th>Metric</th>
<th>RDA</th>
<th>MR-DFS</th>
<th>BoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Total Distance Traveled</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Average Distance Traveled</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Time to Coverage</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Time to Return</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Total Coverage</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Rate of Coverage</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load Balancing</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Idle Time</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 5.1: Common performance metrics for online coverage algorithms.
because communication limitations mean that the return is essential to sharing the information gathered with the human rescuers. Exploration cost corresponds to total distance traveled. Exploration efficiency maps to our metric for rate of coverage, looking at how much the multi-robot algorithm gains in return for what is put in.

Based on common metrics we found in previous research, and the evaluation in [100, 32, 38], we have chosen to use the following metrics for our comparison: total distance traveled, time to coverage, time to return, and rate of coverage. We also discuss scalability of these measures when we add more robots in the more complex environments.

5.3.5 Simulation Results: Structured Environments

We ran our comparison simulations using the Simple Office, NHH and Hospital Section (see Figure 5.13) environments. We used one and three robots in the Simple Office environment (15 runs each); one and four robots in the NHH environment (15 runs each); and one, four, eight, and 12 robots in the Hospital Section environment (15 runs each algorithm/robot combination, with four different starting locations for BoB and RDA-MS).

Both BoB and MR-DFS break down their environments into grids or graphs in which the robots move a uniform distance in a given amount of time. We therefore provide the results from these comparison simulations using a count of the graph vertices visited, rather than actual distances, and time is measured as a single time step to traverse an edge. We set the distances between the vertices in all of the graphs to be roughly equivalent to the communication range in RDA (10 meters), which the physical robots can traverse in approximately 20 seconds.

For the Simple Office environment, we used one and three robots. We show
the rate of coverage for each algorithm in Figure 5.14 and the speedup factor for coverage time in Figure 5.15; provide total distance traveled, time to achieve full coverage, and time until all the robots had returned to their respective starting location(s) in Table 5.2; and show which metrics have statistically significant differences between each pair of algorithms in Table 5.3. For the NHH environment, we used one and four robots. We provide the coverage rate in Figure 5.16, the speedup factor for coverage time in Figure 5.17, the distance and time data in Table 5.4, and the statistical significance data in Table 5.5. For the Hospital Section environment, we used one, four, eight, and 12 robots. We provide the coverage rates in Figures 5.18, 5.19, and 5.20, and the speedup factor for coverage time in Figure 5.21. The distance and time data for all team sizes is shown in Table 5.6, and the statistical significance data in Tables 5.7, 5.8, and 5.9. In looking at the graphs and tables, you can see that BoB and RDA-MS are nearly identical in the smaller, less complex environments (Simple Office and NHH) when using thee or four robots, but diverge in the more complex Hospital Section environment.
Figure 5.14: Rate of coverage for the Simple Office environment, using one and three robots. BoB and RDA-MS have identical rates of coverage. RDA and RDA-EC are identical up to 300 seconds, then diverge.

Figure 5.15: The speedup factor for coverage time with one, two, and four robots with each algorithm in the Simple Office environment.
Table 5.2: Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Simple Office environment using one and three robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots.

<table>
<thead>
<tr>
<th># of Robots</th>
<th>Metric \ Alg.</th>
<th>Travel Distance (edges)</th>
<th>Time to Coverage (seconds)</th>
<th>Time to Return (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BoB</td>
<td>39</td>
<td>660</td>
<td>780</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>45</td>
<td>740</td>
<td>900</td>
</tr>
<tr>
<td>3</td>
<td>BoB</td>
<td>42</td>
<td>180</td>
<td>280</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>63</td>
<td>300</td>
<td>420</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>59</td>
<td>480</td>
<td>600</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>45</td>
<td>180</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>53</td>
<td>480</td>
<td>540</td>
</tr>
</tbody>
</table>

Table 5.3: Significance (p) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Simple Office environment with three robots. NSS means “not statistically significant.”

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
<th>RDA-EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Distance Traveled</td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>NSS</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Coverage</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Return</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
</tbody>
</table>
Figure 5.16: Rate of coverage graphs for the NHH environment, using one and four robots. Note that there was no difference between the algorithms using a single robot in this environment. BoB and RDA-MS have identical rates of coverage.

Table 5.4: Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the NHH environment using one and four robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots.

<table>
<thead>
<tr>
<th># of Robots</th>
<th>Metric \ Alg.</th>
<th>Travel Distance (edges)</th>
<th>Time to Coverage (seconds)</th>
<th>Time to Return (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BoB</td>
<td>41</td>
<td>560</td>
<td>820</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>41</td>
<td>560</td>
<td>820</td>
</tr>
<tr>
<td>4</td>
<td>BoB</td>
<td>46</td>
<td>120</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>60</td>
<td>180</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>64</td>
<td>700</td>
<td>860</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>47</td>
<td><strong>120</strong></td>
<td><strong>220</strong></td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>48</td>
<td>200</td>
<td>340</td>
</tr>
</tbody>
</table>
Figure 5.17: The speedup factor for coverage time with one, two, and four robots with each algorithm in the NHH environment. BoB and RDA-MS have the same speedup factor for each number of robots, so only one line is discernible.

Table 5.5: Significance (p) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the NHH environment with four robots. NSS means “not statistically significant.”

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
<th>RDA-EC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Distance Traveled</strong></td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>NSS</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>NSS</td>
</tr>
<tr>
<td><strong>Time to Coverage</strong></td>
<td>BoB</td>
<td>NSS</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>NSS</td>
</tr>
<tr>
<td><strong>Time to Return</strong></td>
<td>BoB</td>
<td>NSS</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>NSS</td>
</tr>
</tbody>
</table>
Figure 5.18: Rate of coverage for the Hospital Section environment using one and four robots.

Figure 5.19: Rate of coverage for the Hospital Section environment using one and eight robots.
Figure 5.20: Rate of coverage for the Hospital Section environment using one and 12 robots.

Figure 5.21: The speedup factor for coverage time with one, four, eight and 12 robots with each algorithm in the Hospital Section Environment.
Table 5.6: Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Hospital Section environment using one, four, eight, and 12 robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots.

<table>
<thead>
<tr>
<th># of Robots</th>
<th>Metric \ Alg.</th>
<th>Travel Distance (edges)</th>
<th>Time to Coverage (seconds)</th>
<th>Time to Return (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BoB</td>
<td>205</td>
<td>4080</td>
<td>4100</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>259</td>
<td>4860</td>
<td>5180</td>
</tr>
<tr>
<td>4</td>
<td>BoB</td>
<td>238</td>
<td>960</td>
<td>1260</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>396</td>
<td>1420</td>
<td>2080</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>381</td>
<td>2100</td>
<td>2660</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>254</td>
<td>1100</td>
<td>1340</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>324</td>
<td>1500</td>
<td>2020</td>
</tr>
<tr>
<td>8</td>
<td>BoB</td>
<td>225</td>
<td>580</td>
<td>640</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>685</td>
<td>1200</td>
<td>1800</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>486</td>
<td>2160</td>
<td>2580</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>240</td>
<td>540</td>
<td>760</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>528</td>
<td>1100</td>
<td>1620</td>
</tr>
<tr>
<td>12</td>
<td>BoB</td>
<td>239</td>
<td>320</td>
<td>440</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>942</td>
<td>1060</td>
<td>1600</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>733</td>
<td>1740</td>
<td>2300</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>251</td>
<td>380</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>726</td>
<td>1100</td>
<td>1620</td>
</tr>
</tbody>
</table>

The BoB algorithm using a single robot performed slightly better in all measures on the Simple Office and Hospital Section environments, and both BoB and the DFS algorithms performed exactly the same with a single robot on the NHH environment, due to the fact the area for the robots to cover was all narrow corridors, leaving BoB no room to cut corners as it moved to new frontiers. The DFS algorithms run with a single robot are consolidated to a single line in the graphs and tables, because there is no variation between the algorithms when there is only one robot involved (they all perform a basic DFS pattern at that point).
Table 5.7: Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Hospital Section environment with four robots. NSS means “not statistically significant.”

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
<th>RDA-EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Distance Traveled</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>–</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Coverage</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Return</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
</tbody>
</table>

Table 5.8: Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Hospital Section environment with eight robots. NSS means “not statistically significant.”

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
<th>RDA-EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Distance Traveled</td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Coverage</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Return</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
</tbody>
</table>
Table 5.9: Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Hospital Section environment with 12 robots. NSS means “not statistically significant.”

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
<th>RDA-EC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Distance Traveled</strong></td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td><strong>Time to Coverage</strong></td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td><strong>Time to Return</strong></td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
</tr>
</tbody>
</table>

In comparing the performance of the multi-robot runs, we expect BoB and the RDA-MS (multi-start) algorithms to perform the best, because starting in separate locations reduces interference and allows the robots to cover the majority of the environment before interacting with each other (if at all). In the Simple Office and NHH environments, BoB and RDA-MS take the same amount of time to achieve coverage, but RDA-MS has a slightly longer return time and distance traveled because the robots must return along the same path, while BoB has the advantage of a global map and communication so that the robots can plan the shortest path back to their starting locations. However, in the Hospital Section environment, though BoB is best in all categories on 4 robots, the distance traveled and return time are very close to that of RDA-MS, because the BoB robots will travel towards a frontier until they know there isn’t actually more to explore in that area, causing them to cover areas that other robots have already covered as they move towards the frontier, only to get halfway there and then have the other
robots report that the area has been fully covered and there isn’t more for them to do, so they performed a repeated pass on an area unnecessarily. RDA-MS, on the other hand, does not allow the robots to enter an area previously covered by another robot (because the robots are operating as separate teams, there are no calls for additional help), so if the areas are not properly balanced, one robot may end up with a longer path to cover, while the rest of the robots finish early, making the overall distance and return time closer to that of BoB. Our results in the Hospital Section are the average of simulations run 15 times on each of 4 different random starting locations for BoB and RDA-MS. With 8 robots in the Hospital section, we see that RDA-MS still has a longer total distance traveled and return time, but over the 60 runs, averages a slightly faster time to complete coverage. This was due to the robots operating individually and not entering areas covered by other robots, so there was little overlap in initial coverage, which worked out to evenly balance the areas for each robot to work in, even across the different starting locations.

While RDA does not perform well against the comparison algorithms on its own, it does result in lower distances traveled than MR-DFS with the same number of robots in all but the NHH environment. Comparing MR-DFS and RDA-EC (extended communication), however, we can see that the RDA method, when the robots are able to explore larger distances apart, achieves a total distance traveled of about 80% of MR-DFS, even though it takes longer to achieve coverage and return in most cases. In the Hospital Section with 4 robots, though, the RDA-EC algorithm returns all the robots to the starting location faster than the MR-DFS, because the robots explore and return along their paths earlier in the exploration phase, while MR-DFS ends up sending robots to return along side paths to ensure every edge is traversed in each direction, which adds the time after complete
coverage, instead of before. This has the potential to delay getting information
to any rescue workers outside the environment, because the robots don’t return
as quickly. In addition, when using 8 robots in the Hospital Section environment,
RDA-EC performs better than MR-DFS on all measures, because of our focus on
avoiding repeated (and unnecessary) coverage of a given area when at all possible.
Robots running RDA-EC will not move into an already explored area unless a
robot in that area requests additional help because there are multiple branches to
follow or because that robot has reached the edge of the communication range.
While RDA-EC performs, in this case, much worse overall than BoB and RDA-
MS, taking almost twice as long and covering nearly twice as much distance, it is
significantly better than MR-DFS.

With 12 robots in the Hospital Section environment, we don’t see any sig-
nificant improvement in terms of distance traveled for BoB or RDA-MS, and it
continues to increase for MR-DFS, RDA, and RDA-EC. For coverage time, BoB
pulls ahead of RDA-MS again, though not by much, and both algorithms have
faster times to coverage and return. However, RDA-MS suffered here in cover-
age time from there being no very good way to distribute the 12 robots in the
environment. BoB, on the other hand, improved its time, but added distance
compared to the eight robot simulation, as robots far apart from each other both
moved towards frontiers and one would complete the exploration in that area
before the other robot arrived, meaning the second robot traveled that distance
across previously explored areas unnecessarily. The fact that RDA-MS is able to
match and in some areas outperform the BoB algorithm, which requires global
communication and perfect mapping/localization, makes our algorithm more ro-
bust and appealing for use in a disaster scenario, in which communication is often
limited in both range and availability. We also see RDA improving against MR-DFS, especially in terms of distance traveled, though MR-DFS is still significantly faster to complete both the coverage and return. RDA-EC actually performed the same with 12 robots as with eight, due to the manner in which robots are kept at intersections to monitor and direct subsequent and returning robots. Thus, the extra communication range made no difference, because the intersections are all very close together, keeping the robots from spreading out more than they had with eight robots. The robots spent more time waiting to be called to a frontier, so RDA-EC did have a lower total distance traveled than MR-DFS, but that was the only place it performed better.

We have included tables showing the statistical significance $p$ value for the difference in each metric for all pairs of algorithms for each environment and multiple robot combination (see Tables 5.2, 5.4, and 5.6). While statistical significance does not necessarily mean that the differences in distance or time is meaningful, it does give us a clearer picture of whether the differences were achieved due to the algorithm, or just random chance. Our sample sizes are very small, which does mean that a greater difference is required to show statistical significance. The three important items to take away from the tables are i) that RDA-EC and RDA-MS each perform better than RDA in terms of coverage and return time in the complex Hospital Section environment; ii) that RDA-EC performs better than MR-DFS in total distance traveled; iii) and that the difference in performance of RDA-MS and BoB are not statistically significant on any of the three metrics. This last item is the most important, because it shows that, even though BoB has global communication and maps, RDA-MS is able to match its performance with restricted communication (both range and content) and without creating or sharing maps. This makes RDA-MS a more generalizable algorithm for real world
applications, because it doesn’t rely on external communication infrastructure.

We have also provided speedup factor graphs for each environment (see Figures 5.15, 5.17, and 5.21). These show how much the algorithms improve on the time to full coverage as more robots are added. It is uncommon to get super-linear speedup, which is when the factor is greater than the number of robots in use (such as a speedup of 4.25 for four robots over a single robot). However, in this case the reason that we see this in BoB and RDA-MS is because the robots start in separate areas, and do not have to completely explore and then retract along every path, the way the single robot did. That time is instead picked up (to some extent) in the return time. However, because the algorithms have either global communication, or a network by which they can send data to the outside world, we felt that the time to complete coverage was the critical metric, because when that is complete, the information can be provided to the human search and rescue team for their planning.

While BoB was the best overall algorithm, the difference between it and RDA-MS was insignificant. Given the difference in the communication requirements of the two algorithms, we feel this shows that RDA-MS is an appealing approach for use in real world scenarios. RDA-MS also performs better than MR-DFS, especially in terms of distance traveled, which can have a significant impact on the feasibility of using robots for such situations, with their finite energy capacity.

5.3.6 Simulation Results: Unstructured Environments

Since the target application for our algorithms is to aid with search and rescue in disaster scenarios, it is also important to compare the algorithms in unstructured environments, where the obstacles are variable in size, number, and location. We compare the algorithms’ performance in a completely open space, such as
a gym or cafeteria with no obstacles, and in the Cave environment used in the preliminary simulation experiments (see Figure 5.22). In both environments, we ran 15 simulation runs with one, four, and eight robots.

Figure 5.22: Cave environment for simulation experiments.

Results for the Cave environment begin with Table 5.10, which provides the total distance traveled (in edges), and the time to coverage and return (in seconds) for each algorithm. Figures 5.23 and 5.24 then show the rates of coverage for each algorithm using four and eight robots, respectively. The rates of coverage for a single robot using BoB and DFS are provided in both graphs to provide continuity between them. Tables 5.11 and 5.12 provide the statistical significance values for distance traveled and time to coverage and return between each pair of algorithms. Figure 5.25 shows the speedup factor for each algorithm.

Once again, BoB consistently performs the best on all metrics, but its coverage completion time is matched by RDA-MS for both four and eight robots. RDA-MS travels a longer distance and takes slightly longer to return all robots to their starting locations with four robots, but performs exactly the same with
Table 5.10: Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Cave environment using one, four, and eight robots. Values in bold denote the best for that metric for a given number of robots. Values in italics denote when RDA-EC is better than MR-DFS on a particular metric for a given number of robots.

<table>
<thead>
<tr>
<th># of Robots</th>
<th>Metric \ Alg.</th>
<th>Travel Distance (edges)</th>
<th>Time to Coverage (seconds)</th>
<th>Time to Return (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BoB</td>
<td>20</td>
<td>340</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>31</td>
<td>320</td>
<td>600</td>
</tr>
<tr>
<td>4</td>
<td>BoB</td>
<td>24</td>
<td>80</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>51</td>
<td>120</td>
<td>280</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>65</td>
<td>880</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>37</td>
<td>80</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>33</td>
<td>120</td>
<td>240</td>
</tr>
<tr>
<td>8</td>
<td>BoB</td>
<td>28</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>67</td>
<td>80</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>63</td>
<td>360</td>
<td>480</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>28</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>49</td>
<td>140</td>
<td>240</td>
</tr>
</tbody>
</table>

eight robots. In this environment, when starting eight robots in separate locations, they can view half of the environment already. If evenly spaced around the environment, each robot need visit at most two other nodes, and possibly only one other node, so it is a very short coverage time, leaving little opportunity for the algorithms to diverge in performance. Both BoB and RDA-MS improve their coverage and return times with eight robots over the times for four robots, but BoB ends up traveling slightly further with eight robots, while RDA-MS significantly cuts down its travel distance. With more robots, RDA-MS doesn’t have to spend as much time on retracting and then exploring another path around the obstacles, which is where it saves on time and distance.

MR-DFS and RDA-EC perform similarly in terms of coverage and return times
Figure 5.23: Rate of coverage for the Cave environment using one and four robots.

Figure 5.24: Rate of coverage for the Cave environment using one and eight robots.
Figure 5.25: The speedup factor for coverage time with one, four, and eight robots with each algorithm in the Cave Environment.

Table 5.11: Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Cave environment with four robots. NSS means “not statistically significant.”

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
<th>RDA-EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Distance Travelled</td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>NSS</td>
</tr>
<tr>
<td>Time to Coverage</td>
<td>BoB</td>
<td>NSS</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.05</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>NSS</td>
</tr>
<tr>
<td>Time to Return</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
</tbody>
</table>
Table 5.12: Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Cave environment with eight robots. NSS means “not statistically significant.”

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
<th>RDA-EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Distance Traveled</td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>NSS</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Coverage</td>
<td>BoB</td>
<td>NSS</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td>Time to Return</td>
<td>BoB</td>
<td>&lt;0.05</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>&lt;0.05</td>
</tr>
</tbody>
</table>

(differences are not statistically significant), but RDA-EC travels less distance than MR-DFS by a significant margin. However, RDA-EC performs less well with eight robots than with four, because the additional robots add to the time and distance needed for the robots to maneuver around each other from the starting location. Since the environment is made of up neighboring branching vertices, RDA-EC cannot take advantage of the increased communication range, because it still needs to stop robots at the intersections until they are complete, keeping the robots within the original RDA communication range in all but a couple of instances, all of which were made use of with only four robots. So going from four to eight robots was not effective in this environment. Interestingly, RDA significantly improves its time to coverage and return going from four to eight robots, but this has no significant impact on the distance traveled. Instead, more robots just end up acting as sentries for nearly the entire exploration, having moved only one or two vertices from the entrance. The extra robots move the
distance that one of the four robots would have had to traverse multiple times, thus keeping the travel distances similar while reducing the time needed.

In contrast to the large, irregular obstacles in the Cave environment, the Gym environment is a completely open space, 100 meters long by 60 meters wide (roughly the dimensions of an international soccer field). We felt that it would be interesting to see how RDA and its variants perform when there are no obstacles, resulting in all intersections, loops, and dead-ends being defined by proximity to other robots or beacons with different paths to a particular location. MR-DFS is unaffected by this, because it operates only on the graph of the environment. BoB uses the boustrophedon movement patterns, so is also not affected by the lack of obstacles. In fact, BoB has significantly better return times in the Gym environment due to the lack of obstacles. Because RDA returns robots along the same path they took to reach a location, with no obstacles, the time to return is double the time to reach full coverage, which is much less efficient than BoB’s shortest direct path for return.

Results for the Gym environment begin with Table 5.13, which provides the total distance traveled (in edges), and the time to coverage and return (in seconds) for each algorithm. Figures 5.26 and 5.27 then show the rates of coverage for each algorithm using four and eight robots, respectively. The rates of coverage for a single robot using BoB and DFS are provided in both graphs to provide continuity between them. Tables 5.14 and 5.15 provide the statistical significance values for distance traveled and time to coverage and return between each pair of algorithms for four and eight robots, respectively. Figure 5.28 shows the speedup factor for each algorithm.

Due to the need to ensure full coverage, RDA-EC has no advantage over RDA
in the Gym environment, because moving at the very edge of the extended communication range allows the robots to skip portions of the environment, which was an unintended side effect. Therefore, RDA-EC reverts back to RDA in large open spaces, and only improves in performance if there are also smaller corridors or spaces that allow the robots to move apart without missing any areas in the sensor coverage. In the result graphs and tables, we report only RDA values, and omit RDA-EC performance values, as they are identical in all cases.

We get very different exploration patterns between BoB, MR-DFS, RDA-MS, and RDA. We can depict the Gym environment as a graph of 10 nodes by 6 nodes, evenly spaced in a grid pattern, as in Figure 5.29. Using a single robot, the exploration pattern is shown for BoB in Figure 5.30 and in Figure 5.31 for the DFS algorithms. As expected, BoB explores in a back and forth pattern, and the DFS algorithms explore in a spiral. For the DFS algorithms, the space where

<table>
<thead>
<tr>
<th># of Robots</th>
<th>Metric \ Alg.</th>
<th>Travel Distance (edges)</th>
<th>Time to Coverage (seconds)</th>
<th>Time to Return (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BoB</td>
<td>70</td>
<td>1200</td>
<td>1300</td>
</tr>
<tr>
<td></td>
<td>DFS</td>
<td>120</td>
<td>1200</td>
<td>2400</td>
</tr>
<tr>
<td>4</td>
<td>BoB</td>
<td>80</td>
<td>300</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>296</td>
<td>1000</td>
<td>1280</td>
</tr>
<tr>
<td></td>
<td>RDA &amp; RDA-EC</td>
<td>496</td>
<td>4520</td>
<td>5800</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>120</td>
<td>300</td>
<td>600</td>
</tr>
<tr>
<td>76</td>
<td>BoB</td>
<td>8</td>
<td>160</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>256</td>
<td>480</td>
<td>780</td>
</tr>
<tr>
<td></td>
<td>RDA &amp; RDA-EC</td>
<td>898</td>
<td>8300</td>
<td>10580</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>120</td>
<td>160</td>
<td>320</td>
</tr>
</tbody>
</table>

Table 5.13: Travel distance in number of edges traversed, and time to coverage and return in seconds, for each algorithm in the Gym environment using one, four, and eight robots. Values in bold denote the best for that metric for a given number of robots.
Figure 5.26: Rate of coverage for the Gym environment using one and four robots. BoB and DFS with one robot have identical rates, and with four robots BoB and RDA-MS also have identical rates of coverage.

Figure 5.27: Rate of coverage for the Gym environment using one and eight robots. BoB and DFS with one robot have identical rates, as do BoB and RDA-MS with eight. RDA’s rate of coverage follows the same step pattern to 100% (not shown).
Figure 5.28: The speedup factor for coverage time with one, four, and eight robots with each algorithm in the Gym Environment. BoB and RDA-MS have identical speed up lines.

Table 5.14: Significance ($p$) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Gym environment with four robots. NSS means "not statistically significant."

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Distance Traveled</td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Time to Coverage</td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Time to Return</td>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
</tr>
<tr>
<td></td>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
</tr>
</tbody>
</table>
Table 5.15: Significance \( (p) \) values for each pair of algorithms for the metrics total distance traveled, time to coverage, and time to return in the Gym environment with eight robots. NSS means "not statistically significant."

<table>
<thead>
<tr>
<th>Metric</th>
<th>Algorithm</th>
<th>MR-DFS</th>
<th>RDA</th>
<th>RDA-MS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Distance Traveled</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td></td>
</tr>
<tr>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td></td>
</tr>
<tr>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td></td>
</tr>
<tr>
<td><strong>Time to Coverage</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>NSS</td>
<td></td>
</tr>
<tr>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td></td>
</tr>
<tr>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td></td>
</tr>
<tr>
<td><strong>Time to Return</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoB</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.05</td>
<td></td>
</tr>
<tr>
<td>MR-DFS</td>
<td>–</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td></td>
</tr>
<tr>
<td>RDA</td>
<td>–</td>
<td>–</td>
<td>&lt;0.01</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.29: Gym environment with all nodes and possible edges.

there were edges in the original graph, but which are empty in the single robot exploration are spaces where loops were discovered, and the robot returned to the path it had been following, which is what causes the spiral pattern. For MR-DFS, this also serves to mark every edge as covered in both directions, though
Figure 5.30: BoB exploration path in the Gym environment with a single robot from the starting node (S) to the node where full coverage is achieved (C).

Figure 5.31: DFS exploration path in the Gym environment with a single robot from the starting node (S) to the node where full coverage is achieved (C).
Figure 5.32: BoB exploration path in the Gym environment with four robots from the starting nodes (S) to the nodes where full coverage is achieved (C). Nodes covered by a specific robot are connected by arrows depicting the path the robot took, and are separate from the areas covered by other robots.

Figure 5.33: RDA-MS exploration path in the Gym environment with four robots from the starting nodes (S) to the nodes where full coverage is achieved (C). Nodes covered by a specific robot are connected by arrows depicting the path the robot took, and are separate from the areas covered by other robots.
it does mean that MR-DFS has the same long return path as RDA. The Gym environment highlights the observation made previously, that RDA is not efficient when presented with single, long paths, because of the manner in which the robots move around each other. Because RDA-MS puts each robot on a separate team when there are eight or fewer robots, it is able to avoid this inefficiency, but if instead there were multiple teams of two or four robots, we would see similar slowdowns in this type of environment.

The patterns are very similar with four robots for BoB (see Figure 5.32) and RDA-MS (see Figure 5.33), where the improvement of having four robots is obvious by the shorter paths for each robot. RDA, however, shows how having to maneuver around every other robot in order to reach the frontier along a single path slows the coverage and return (see Figure 5.34). MR-DFS has yet a different pattern with four robots, in which the robots end up creating a criss-crossing...
pattern, which also serves to achieve the goal of traveling every edge in both directions, but which also means a lot of extra distance traveled overall. As with the Cave environment, adding robots to the exploration leads to significant improvement for BoB and RDA-MS, which are still performing at the same level, but additional robots lead to a decrease in performance for RDA and MR-DFS. Thus, it would be inadvisable to use RDA or MR-DFS in an environment with large open spaces.

Overall, BoB and RDA-MS perform identically in time to full coverage, nearly the same in distance, but RDA-MS falls behind in the return time, due to the manner in which the robots return. This is the area where BoB’s global communication is particularly useful, because it allows the robots to plot and take the shortest (and thus fastest) route back to the starting location. However, since the exploration is completed in the same amount of time, RDA-MS is still an effective alternative, without the global communication requirement. One additional feature in the Gym environment is that RDA-MS requires exactly the total distance traveled no matter how many robots are deployed, which would allow for precise planning of how many robots would be needed based on energy consumption. This is not likely to be a common scenario, however, as most environments have at least some obstacles.

5.3.7 Physical Robot Experiment Results

We also performed physical robot experiments using RDA, RDA-MS, and RDA-EC in the NHH environment. For these experiments, we used the LEGO EV3 robots, running the ev3dev OS, as described previously in Section 5.2.2.

Our experiments consisted of 10 runs each using one, two, and three robots for each of the algorithms, though for RDA-MS, that ended up being five runs
Table 5.16: Travel distance in meters, and time to coverage and return in seconds, for each algorithm and number of robots combination run. Values in bold denote best for that metric for a given number of robots.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Metric \ Alg.</th>
<th>Travel Distance (meters)</th>
<th>Time to Coverage (seconds)</th>
<th>Time to Return (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NHH</td>
<td># of Robots</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DFS</td>
<td>123</td>
<td>1588</td>
<td>2074</td>
</tr>
<tr>
<td></td>
<td>RDA</td>
<td>180</td>
<td>1502</td>
<td>2226</td>
</tr>
<tr>
<td>2</td>
<td>RDA-MS</td>
<td>120</td>
<td>616</td>
<td>1082</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>150</td>
<td>720</td>
<td>1381</td>
</tr>
<tr>
<td>3</td>
<td>RDA</td>
<td>207</td>
<td>942</td>
<td>1632</td>
</tr>
<tr>
<td></td>
<td>RDA-MS</td>
<td>117</td>
<td>454</td>
<td>744</td>
</tr>
<tr>
<td></td>
<td>RDA-EC</td>
<td>135</td>
<td>498</td>
<td>856</td>
</tr>
</tbody>
</table>

for each of two starting configurations with two and three robots. We measured the total distance traveled (sum of distance traveled by each robot, in meters), as well as the time (in seconds) to coverage and return, all shown in Table 5.16. As expected, based on the simulation results, RDA-MS performed the best and RDA the worst on all metrics. We see improved rate of coverage as the number of robots increases (see Fig. 5.35).

In RDA, as the number of robots increases, the number and length of plateaus in the rate of coverage also increases, due to the need to fully return along a path before moving down a different branch. We do not see this as much in RDA-EC, due to the longer communication range, which allows the robots to move further apart. RDA-MS shows none of this because the robots are completely independent. With larger environments and more robots, we expect to see more plateaus. We would also expect to see more plateaus in environments with few obstacles, as seen in the RDA results in the Gym environment simulations.
Figure 5.35: Results of running one, two, and three Lego EV3 robots using RDA, RDA-EC, and RDA-MS in the NHH environment.
Chapter 6

Conclusions and Future Work

We have investigated the problem of using a multi-robot team to explore and provide full coverage of an enclosed but unknown environment. We examined the previous work in the area of robot exploration and coverage, both for single and multiple robot systems, paying particular attention to feasibility for real world applications of the systems. In narrowing our target scenarios to disaster or search and rescue situations, we added the complexity of limited communication, which led us to pursue distributed rather than centralized systems. The availability of robots, and the specific hardware of the robots in our target scenario is unknown, and may vary from one situation to the next, so we also focused on developing algorithms that will work for robots with limited sensing and actuation capabilities, and with a small team, including just a single robot.

We then developed two new distributed algorithms, the Rolling Dispersion Algorithm (RDA) and the Sweep Exploration Algorithm (SEA), for online multi-robot exploration. Both algorithms operate under significant communication restrictions, similar to those that may be encountered in actual disaster environments as well as remote locations. The algorithms also function on very basic...
robots, making it relatively cost efficient for real world implementation. Theoretical proofs of correctness show that the algorithms are both guaranteed to complete the exploration, with minimal duplicated exploration and no infinite loops, and return to the starting location, even with robot and beacon attrition. The minimum requirements for the algorithms to function are a single robot with obstacle detection and a means of communicating, and beacons to distribute in the environment. While the number of beacons necessary may be large (and unknown prior to the exploration), they can be simple motes that sit on a button battery, making them both inexpensive and small enough to be carried in significant quantities.

We provide analysis showing that SEA uses the minimum number of message types in order to complete the exploration, while providing guarantees of avoiding repeated exploration of an area, avoiding loops, accommodating robot or beacon failures, and still providing full coverage and returning the functional robots to the entry location when the exploration is complete. We also provide analysis of the fault tolerance in terms of added exploration time for recovery under the possible failure conditions.

In addition to the theoretical analysis, we have provided simulation and physical robot experimental results for both algorithms, and discussed which algorithm performs best in different types of environments. While SEA completes the exploration, it is very slow to do so, making it less useful for search and rescue situations, in which speed is critical. However, there are some layouts in which it performs the exploration more quickly than RDA, due to the more rigid manner in which SEA explores. In most cases, RDA is the better algorithm, being faster to achieve full coverage and more capable of sharing pertinent information during and after the exploration.
We have also compared RDA with two similar algorithms, MR-DFS and BoB, from other research groups. We developed two minor modifications to RDA, RDA-EC (extended coverage) to compare with MR-DFS, and RDA-MS (multi-start) to compare with BoB. We created these modifications to incorporate some of the features we saw in the other algorithms, and to put our algorithm on more even footing while maintaining our communication restrictions, which were not a part of either MR-DFS or BoB. In the comparison simulation results, we showed that the modified versions of RDA perform on par with their respective comparison algorithms, and, as we increase the number of robots, we found that RDA-MS is able to complete the coverage faster than BoB. Without the need for global communication that is present in BoB, this makes RDA-MS an appealing option for search and rescue scenarios.

6.1 Limitations and Future Work

The biggest limitation of our algorithms, both RDA and SEA, is the speed with which they explore and achieve full coverage of the environment. This is primarily restricted by the manner in which robots move from one area to a frontier to continue the exploration. If more robots are required at one frontier, a robot from the edge of the team (acting as sentry for other robots, but with only beacons as its sentries) drops an unexplored beacon to mark the path, and then follows the call path out to the frontier. This means that, for a path length of $pl$ with $n$ robots along it, the call must make $n$ hops to reach the new explorer, and that new explorer has to travel $pl$, maneuvering around the $n$ robots, to reach the frontier. This can be a very slow process, and is exacerbated in SEA when only one robot can be moving at a time. Thus, the primary avenue of future work that we are
considering is to implement alternative movement patterns for the robots.

The first alternate movement pattern would operate much like a train, in which each robot acts as a train car, being pulled along by the robot in front of it through their efforts to maintain communication at all times. At a branch, one robot would split the train of robots between the available paths, creating new "engines" for each path to continue pulling the robots along. When a train reaches a dead-end, the engine would apply a push force instead of a pull, and the robot at the other end of the train (or closest to the branch, if there are robots down multiple paths) would become the engine and pull the train out to the next frontier they need to explore. This could provide us with a more balanced use of power, by reversing which robots get out to the end of the paths, but could also put the beacon distribution out of balance, if the last robot to return along a path has to place all of the explored beacons. We believe there are ways to balance the beacon distribution back out, and the potential improvement in exploration speed is significant and would outweigh the imbalance of beacon distribution.

Our second alternate movement pattern would actually be to relax the communication restriction, and allow the robots to move out of range of each other. While this is similar to MR-DFS, the major difference would be that the robots would still be placing beacons each time they reach the edge of the communication range, leaving the full network of beacons in place as the robots conduct the exploration. This method would allow all of the robots to continue moving during the exploration, which allows them to reach new frontiers much more quickly. However, in some environments, such as the NHH environment, with few branches and long narrow corridors, the robots will end up simply covering the same area one after another as they search for new frontiers. With the beacon network, we can reduce this redundancy by having the robots wait at an intersection until a
robot on one of those paths announces that it has another branch. Then only a limited number of additional robots would traverse the path again, and hopefully the first robot won’t complete all of the other paths in the time it takes the robots to get to the next intersection.

A secondary limitation in our experiments is that we have only tested ground robots on level surfaces with no difficult obstacles (chairs, tables, trash cans, or other small or movable items). This means that our algorithms are untested in environments we would encounter in a disaster or search and rescue situation. While we have designed our algorithms to function in those types of scenarios, and we have shown that the communication restrictions lead to complete coverage of the accessible area in a static, easy environment, we see potential challenges when the environment is less easy to move about in. So we feel that additional physical robot experiments, using robots that would be appropriate for search and rescue (or at least approximate such robots), are still needed. It will also be important to consider what will happen if there are robots on multiple levels/floors of an environment, and how that will impact the detection of loops. It may be that the robots will need to confirm that they can reach another robot or beacon before determining that they have found a loop and blocking it off.

Lastly, we have worked only with heterogeneous robots, but it is now very common that a robot team will be made up of multiple types of robots. This would also make the system more appealing for disaster scenarios, as there are instances where an aerial robot, or a small insect-like robot that can fit into smaller spaces, would be able to get somewhere that our ground robots cannot. Having multiple types of ground robots is also advantageous, as each robot may have different skills that can be applied to the situation. We do not anticipate any issues in using our algorithms with heterogeneous teams, but we can see that the
algorithms might need some adjustment to fully utilize different types of robots. Developing these aspects of the system would also lead to investigating how to calculate or estimate the required or optimal number of robots for a given scenario.
References


