Scholars' Mine

Doctoral Dissertations

Student Theses and Dissertations

Spring 2014

Foundations of coverage algorithms in autonomic mobile sensor networks

Mark Edward Snyder

Follow this and additional works at: https://scholarsmine.mst.edu/doctoral_dissertations

Part of the Computer Sciences Commons

Department: Computer Science

Recommended Citation

FOUNDATIONS OF COVERAGE ALGORITHMS IN AUTONOMIC MOBILE

SENSOR NETWORKS

by

MARK EDWARD SNYDER

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2014

Approved by

Dr. Sriram Chellappan, Advisor
Dr. Dan Lin
Dr. Bruce McMillin
Dr. Chaman Sabharwal
Dr. Mayur Thakur

PUBLICATION DISSERTATION OPTION

This dissertation has been prepared from a body of work in the field of area coverage algorithms for mobile sensor networks and has been prepared from the Missouri University of Science and Technology specifications.

Paper 1: Pages 9–30 have been published as *Event Coverage in Sparse Mobile Sensor Networks*, published in Proceedings of The 12th International Conference on Network-Based Information Systems (NBIS), Indianapolis, Indiana, August 2009 with Sriram Chellappan.

Paper 2: Pages 31–54 have been published as *Exploratory Coverage in Limited Mobility Sensor Networks*, published in Proceedings of The 16th International Conference on Network-Based Information Systems (NBIS), Guangju, Korea, September 2013 with Sriram Chellappan and Mayur Thakur, and has been accepted, in extended form, for publication as *Distributed Exploratory Coverage with Limited Mobility* to International Journal of Space-Based and Situated Computing (IJSSC) with Sriram Chellappan and Mayur Thakur.

Paper 3: Pages 55–71 have been published as *Fault Tolerance in Area Coverage Algorithms for Limited Mobility Sensor Networks*, published in Proceedings of The Thirteenth International Conference on Networks (ICN), Nice, France, February 2014 with Sriram Chellappan.

ABSTRACT

Drones are poised to become a prominent focus of advances in the near future as hardware platforms manufactured via mass production become accessible to consumers in higher quantities at lower costs than ever before. As more ways to utilize such devices become more popular, algorithms for directing the activities of mobile sensors must expand in order to automate their work.

This work explores algorithms used to direct the behavior of networks of autonomous mobile sensors, and in particular how such networks can operate to achieve coverage of a field using mobility. We focus special attention to the way limited mobility affects the performance (and other factors) of algorithms traditionally applied to area coverage and event detection problems.

Strategies for maximizing event detection and minimizing detection delay as mobile sensors with limited mobility are explored in the first part of this work. Next we examine exploratory coverage, a new way of analyzing sensor coverage, concerned more with covering each part of the coverage field once, while minimizing mobility required to achieve this level of 1-coverage. This analysis is contained in the second part of this work.

Extending the analysis of mobility, we next strive to explore the novel topic of disabled mobility in mobile sensors, and how algorithms might react to increase effectiveness given that some sensors have lost mobility while retaining other senses. This work analyzes algorithm effectiveness in light of disabled mobility, demonstrates how this particular failure mode impacts common coverage algorithms, and presents ways to adjust algorithms to mitigate performance losses.

ACKNOWLEDGMENTS

There are many people to thank, notably Dr. Sriram Chellappan for all his years of support as my academic advisor, Dr. Bruce McMillin for his support and encouragement, Dr. Mayur Thakur for inspiration and guidance, and Dr. Chaman Sabharwal and Dr. Dan Lin for evaluating my work. In many ways each of us are the product of our parents, and I believe I must acknowledge the influence mine have had on my life. My father Mackey Snyder showed me a consistent example of an honorable life, being true to others even when nobody was watching. His work ethic has always been something I have admired greatly, working an assembly line job to make ends meet, then sacrificing his free time to build a lasting legacy when many of his peers were living for the moment. As I watch my father rise and work hard to accomplish something productive each day, I realize where I get the ability to buckle down and grind through problems that can't be solved quickly. My mother Carol (Glass) Snyder has always been the great complement to my father. Her creativity, grace, beauty, positive attitude, and ability to be flexible as counterpoint to my father's uniformity shows me where I get the ability to work with others in a team and to approach people from a position of mutual respect and humility. I have also been truly blessed by my wife Sandra (Kolb) Snyder who rounds out the rough edges where my parents' qualities in me combine in awkward ways, and the improvements in my character are due to her love, persistence, and patience with me over these past two decades together. My success in this life has been directly proportional to how I have been true to the highest ideals these individuals have exemplified. My failures all examples of reaching for lesser virtues. I cannot merely thank these individuals for contributing to my success, but rather my success is theirs, and I hope that I have honored them through this work.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

PAPER III

SECTION

LIST OF TABLES

1. INTRODUCTION

This work explores the rich field of study around the effects of limited mobility with networks of mobile sensors, and how these constraints impact the effectiveness of coverage algorithms. As the application of autonomous drones becomes more approachable–platforms have recently become available via retail channels to consumers–the algorithms to drive autonomous behavior of these platforms will be ever more important.

1.1. PLATFORMS AND APPLICATIONS

Security applications (such as border monitoring), search and rescue, marine habitat preservation, livestock management–the list of possible applications for autonomous mobile sensor networks abounds.

A common theme in these applications is that a network of mobile sensors, acting autonomously, can provide tremendous value if it can efficiently and effectively cover a search area. The area of particular interest in this work is how limited mobility affects the performance of mobile sensors.

A network of sensors can be used to monitor the environment, for example, monitoring for forest fires, tsunamis, or other natural phenomenon. A network of sensors can monitor a larger area with less human intervention and higher accuracy because sensors are becoming more economical. Effective algorithms employed on these platforms can extend the lifetime of networks of sensors, improve efficiency and effectiveness of sensors at meeting Quality of Service (QoS) goals, and overcome impediments that historically have made these platforms less than useful or even impractical.
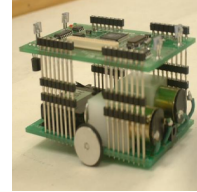
Figure 1.1. The DraganFlyer™X6 UAV

Research into advances for robotic platforms is occurring on many fronts [1]. UAV Platforms such as the DraganFlyer™X6 [2] as seen in Figure 1.1 show great promise as both an academic as well as practical model in real-world commercial applications. Less recently, numerous efforts have focused on building self powered miniature mobile robot sensor platforms as shown in Table 1.1. The Robomote (70mm x 45mm x 35mm) [3], Khepera series of robots (70mm x 30mm) [4, 5], and the XYZ platform [6] are prototypes of mobile sensors that are battery powered and motor driven. In each of these platforms, sensors (to sense events) and communication devices create mobile communicating sensors. A rather novel approach to mobility in sensor platforms utilizes a fuel-powered hopping mechanism [7] used in Intelligent Mobile Land Mine Units (IMLM) developed by DARPA.

The fact that these platforms are subject to such limitations validates the importance of analysis of the algorithms governing their behavior. Recently, additional platforms have emerged and continue to build in popularity as the cost declines. Still, limits on the mobility of these platforms demands improved algorithmic support.

Table 1.1. Practical Implementations of Mobile Sensors

	Khepera III	**XYZ**	**Robomote**	**IMLM**
Mobility Source	DC brushed motors	Miniature geared motor	DC motors	Fuel powered ignition
Movement Type	Continuous	Continuous	Continuous	Hops, Flip-based
Mobility Distance	3600m	165m	360m	100 hops/10m
Manufacturer	K-Team	Yale University	USC	DARPA

Even the classic vacuuming robot, which several popular brands have made common in modern households, is an example of an autonomous mobile sensor platform attached to a household device (see Table 1.2). Features include scheduled activation, independent algorithms for discovery and navigation of their environment, and the ability to sustain extended operation by self-guidance back to the charging base. These devices garner high customer satisfaction through successful utilization of coverage algorithms, and have become a common, trusted, technology for everyday usage. Likewise, as the affordability, practicality, and technical capabilities of mobile sensors increases at the same time that the cost, utility, effective lifetime, and sustainable operation becomes more mature, networks of mobile sensors will continue to gain more and more popularity and significance in our society.

Application development is still one of the main hurdles to wide adoption of autonomous mobile sensors [8]. Previously, algorithm developers could not rely on a

Table 1.2. Vacuuming Robots

iRobot Roomba	Neato Robotics XV-11

progression of abstraction layers between the application and the low level operating system and concerns of hardware control. Even now, substantial knowledge of the intricacies of hardware concerns is required to program algorithms for the simplest of mobile sensor coverage algorithms.

1.2. PROBLEMS POSED BY LIMITED MOBILITY

Three important classifications of coverage algorithms include Blanket, Sweep, and Barrier Coverage [9], as well as Exploratory Coverage, the focus of Paper 2. Another way of categorizing coverage algorithms is whether the goal is Area Coverage or Event Coverage. We describe Area Coverage as seeking to arrange sensors to cover (or "sense") every point in a search area, with several modalities. First, 1-coverage has been narrowly referred to as an arrangement of sensors such that all points in the search area are within sensor range of at least one mobile sensor. However, we broadly refer to 1-coverage as sensors using a mobility strategy to pass over all points in the search area after a period of time. The narrower definition is a special case where the time period is 0. We will call this simultaneous 1-coverage. Simultaneous n-coverage is described as covering all points at the same (or for a period of) time by

n sensors. Next, we use the term polling to refer to sensors passing over all points with a certain frequency during a period of time.

In these problems, as with other applications involving sensor mobility, a number of key coverage metrics are impacted when mobility restrictions are introduced and so we say that limited mobility affects the performance of individual sensors and networks of mobile sensors in a variety of ways.

With the number of sensors is sufficient and mobility is unlimited a group of sensors can eventually achieve 1-coverage. Obviously, strategies can be designed that will not achieve the goal, such as sitting in one place or moving in a circle forever. However, even given an inefficient strategy such as moving in random directions for random periods of time, the sensors will pass over all points in the search area eventually. If there are m sensors and n tiles in a coverage area represented as a grid, and sensors can travel to (and cover) 1 new tile each time cycle, then it will take at least n/m cycles to fully cover the area once. If the total number of navigation opportunities is less than n/m then we would say that 1-coverage cannot be achieved. However, there are many scenarios where the navigation opportunities available are greater than n/m, but 1-coverage still cannot be achieved. For example, in the above scenario if the number of navigation opportunities is exactly n/m and any sensor moves to a tile that was already covered by another sensor, then that move was wasted and 1-coverage can no longer be achieved. Further, in some scenarios, an algorithm may assure us that it can guarantee a solution that achieves 1-coverage, but it may do so in a suboptimal fashion in which case we seek better algorithms that can cover the same area using less time or less mobility.

The speed at which events in a search space can be detected is also subject to the limitations on mobility. The time required to achieve 1-coverage from an initial deployment affects event detection, but in a sparse deployment of sensors where 1-coverage is achieved but not maintained, the time interval required for the network

to re-achieve 1-coverage again (or to poll the region) is also a factor that is dependent upon mobility limitations.

The lifetime of a network of mobile sensors is another area where mobility plays a key part. Some algorithms are designed to allow sensors to spread as evenly as possible over a region using virtual forces [10]. If a sensor were to fail, the network's ability to provide blanket coverage can be extended by the fact that sensors may reposition themselves to cover the hole–the area of which no sensor is within range. Sensors may continue to fail until the sum of area covered by sensors is exceeded by the area of the region. In addition to the way sensor failure affects the lifetime of the network, placing a constraint on the mobility of the sensors affects lifetime further, in that once all mobility is expended in the activity, the sensors might no longer be able to reposition themselves and the network would no longer meet the Quality of Service (QoS) metric imposed by the problem–the ability to maintain blanket coverage of the entire area–thus the network would fail prematurely. Once again, we see that limitations on mobility affect a key factor in a problem that utilizes mobile sensors.

1.3. IMPROVING PERFORMANCE GIVEN LIMITED MOBILITY

In Paper 1 we examine a surveillance problem wherein mobile sensors deployed in a sparse configuration are expected to detect events of interest in a search field, where the location and duration of each event in the network is unknown. A movement algorithm based on efficient traversal of a countour (or tour) for the sensors in the network is designed to meet two objectives. The first is maximizing event detection and the second is minimizing detection delay. The performance of our algorithm is demonstrated from the perspective of event detection and delay with respect to the number of sensors, movement velocity, and the number and duration of events.

1.4. EXPLORATORY COVERAGE USING LIMITED MOBILITY SENSORS

In Paper 2 we examine practical algorithms for employing mobile sensors deployed as a network of unmanned, autonomous devices with limited mobility. Once again, we focused on sparse deployment configurations and problem scenarios where it is crucial that events are located and identified quickly using a new type of problem called exploratory coverage. We also defined a taxonomy of the various subproblems within this problem space as a tool to aid in further study of the problem. We then demonstrated the performance of algorithms designed for exploratory coverage to show that the effects of mobility constraints vary the level of coverage achieved more in sparse deployments than when sensors are deployed in a more concentrated fashion.

1.5. FAULT TOLERANCE IN AREA COVERAGE ALGORITHMS FOR LIMITED MOBILITY SENSOR NETWORKS

In Paper 3 we study fault tolerance and the effects of disabled mobility on coverage algorithms. In this context, we define the concept of disabled to mean that failure mode has occurred in which a mobile sensor has experienced some condition where it retains all functions except its ability to reposition itself. We introduce a schedule function for the loss of mobility for deployed sensors so that gradually more and more sensors become disabled over time. At any single point in time, the network can be viewed as a hybrid sensor network in which some sensors are fixed in their deployed position while others are mobile. Although this is analogous to the problem of disabled mobility, the fact that sensors unexpectedly lose their mobility at a random yet statistically predictable rate simulates a real world phenomenon and we seek to study the effects of this dynamic on various algorithms. Given some basic objective measurements for the performance of mobile sensors, we learn about the

effects of disabled mobility on various algorithms for mobile sensor coverage in the light of disabled mobility. Of interest are features of algorithms that may make them particularly suited to address (either mitigate or prevent) the specific performance limitations of the network under such disabilities. We also seek to discover adjustments that can show resulting improvements on performance measurements when faced with disabled mobility.

We observe that coverage algorithms that do a good job of quickly reaching a desirable location from their initial deployment, cooperate to avoid gaps and redundant coverage, and continue to leverage what mobility is available throughout the sensor network, produce better results as sensors lose their mobility than algorithms that rely on statically touring or other more methodical means of exploring and covering their environment.

With extremely sparse deployments, sensors that are mobile come into contact with disabled sensors less often. In these scenarios, we observe that algorithms such as Random Direction Walk have less of an impact than algorithms that tour an established territory, because in the latter case, once a sensor becomes disabled, there is no sensor to cover that sensor's territory. As the deployment becomes less sparse, algorithms that try to avoid one another are more vulnerable to being misled by disabled sensors that continue to broadcast their intentions to move, but never do.

PAPER I. EVENT COVERAGE IN SPARSE MOBILE SENSOR NETWORKS

1. ABSTRACT

Autonomous mobile sensors are employed with ever-increasing frequency, in applications ranging from search and rescue, detection of forest fires, and battlefield surveillance. In this paper, we consider a representative surveillance problem wherein a sparse number of mobile sensors are expected to cover events of interest in a deployment field. Each event appears for a certain time and then disappears. Furthermore, the location of each event and its duration is unknown. In this paper, we design a sensor movement strategy based on efficient traversal in the network to fulfill two objectives: maximizing event detection, and minimizing detection delay. Analysis and simulations demonstrate the performance of our algorithm from the perspective of event detection and delay with respect to the number of sensors, movement velocity, and the number and duration of events.

2. INTRODUCTION

Mobility in wireless sensor networks is a topic that has received significant attention lately. A host of surveillance missions today can significantly benefit from mobile sensors. Instances include battlefield missions like intruder tracking, military missions like monitoring forest fires in a large scale area, homeland security missions like anthrax/ poison/ explosive sensing. Traditionally, the mobile entities in such missions have always involved human beings, where a critical issue becomes the safety of the entities. In many instances, soldiers have been fatally shot in searching a suspect hide-out; bomb squads face great risk when searching for explosives; fire

personnel have been killed en masse in disaster recovery operations (e.g., 9/11). A major motivation for the investment in mobile sensors is to replace the human element in such missions with a view to minimize casualties.

In this paper, we consider the following scenario. A set of events occur in a deployment field that must be covered (i.e., sensed). The events are dynamic in the sense that each event initially appears and then fades away after a certain time. An event is said to be *live* between the time the event has started to appear and the time it fades away. An event is *covered* if it is within the sensing range of at least one sensor while the event is still live. Events that are not covered while they are live can never be subsequently sensed. Such events are considered *lost*. The number of events, their locations, and their durations are all unknown a priori. In this scenario, we address the following problem: Given a sparse deployment of mobile sensors in the deployment field, we want to design a movement strategy for the sensors that maximizes the number of events covered, while minimizing the detection delay.

Given the lack of specific information on distribution of event occurrence locations, the best strategy for the mobile sensors is to continuously traverse the entire network area monitoring for events. In this paper, we propose a simple and yet highly efficient movement strategy for sensors to traverse the network area. The efficiency is gauged based on minimizing the overall movement distance traversed by sensors, while still maximizing area covered. To this end, we first compute the corresponding contour for the sensors to traverse. Then, we develop a strategy for traversing this contour in the presence of multiple sensors. For this contour, we then derive analytical bounds, and also perform extensive simulations to confirm the calculated probability of event coverage and event detection delay with respect to the number of sensors, movement velocity and the number and duration of events. Our analysis reveals that our proposed movement algorithm can achieve a high probability of event coverage while also reducing the associated event detection delay.

3. RELATED WORK

The first major significance of sensor mobility was realized in disaster recovery operations post 9/11, when Dr. Robin Murphy, and her colleagues at The Center for Robot-Assisted Search and Rescue in Tampa, Florida flew to Ground Zero with a swarm of self powered tiny mobile robotic camera sensors (as small as 17cm x 6cm x 5cm) to explore the debris and search for potential survivors [11]. In fact, this was the first known major operation using mobile sensors for urban search and rescue. With the success of this mission, numerous activities both in the academia and the industry have subsequently ensued due to the tremendous advantage mobile sensors can offer today in a wide spectrum of missions. In the following, we present some important related work in two areas: Practical mobile sensor implementations and Theoretical research on algorithm design for sensor mobility assisted coverage.

3.1. Practical Implementations of Mobile Sensors. In the recent past, numerous efforts have focused on building self-powered miniature mobile robot sensor platforms as shown in Figure 1.1 on Page 3. The Robomote (70mm x 45mm x 35mm) [3], Khepera (70mm x 30mm) [4], and the XYZ platform [6] are prototypes of mobile sensors that are battery powered and motor driven. In each of these prototypes, appropriate sensors (to sense events), and communication devices are provisioned in order to realize mobile communicating sensors. Needless to say, the realization of such types of sensor designs is primarily tuned towards elimination of the human component and associated casualties in critical surveillance missions where dangers are likely. The mobility algorithm we design in this paper is primarily targeted at being executed by such sensors.

3.2. Mobility Assisted Coverage in Sensor Networks. In the recent past, a host of research activities has focused on how to exploit sensor mobility for coverage enhancement. Such work can be broadly classified into two: area coverage

and event coverage. In the following, we present important related work on sensor mobility algorithms design for enhancing each of the two classes of coverage.

Area Coverage. In this class of movement algorithms, the objective is to move sensors starting from initial arbitrary positions to desired final positions such that at those final positions (i.e., at the conclusion of sensor movements), the percentage of area covered by sensors is maximized. An important ancillary objective is to minimize the number of sensor movements in achieving the coverage objective.

Wang, Cao and La Porta in [10] design a virtual force algorithm for improving 1-coverage in the network after an initial random deployment of sensors. We use the term 1-coverage to means that all areas in the network need to be within the sensing range of at least one sensor. In their algorithm, sensors exert virtual forces, where closer sensors repel one other and farther sensor attract. In this manner, the sensors spread themselves uniformly in the network after several iterations. Heuristics are proposed to minimize unnecessary movements. In [12] the virtual force approach is extended to consider coverage and secure connectivity, where weights are assigned to forces depending on keys shared.

In [13], Liu, Brass, Dousse et. al., propose a model where each sensor independently moves by choosing a random direction and speed, for a particular interval. After each interval, a new direction and speed are randomly chosen, and the process repeats for the duration of the mission. The authors analytically demonstrate that while the fraction of area covered at any instant remains unchanged, the total area covered during a time interval significantly improves in this mobility model. Our algorithm improves on this idea by proposing event duration and modification that produces an highly efficient coverage strategy.

In [14, 15, 16], the authors consider optimum area coverage with limited mobility sensors. In [14, 15], centralized optimum algorithms are proposed for limited

mobility sensors to achieve 1-coverage and k-coverage, respectively. In [16], distributed algorithms are proposed for k-coverage, along with bounds on number of sensors needed for coverage.

Event Coverage. Butler and Rus in [17] design Voronoi diagram-based algorithms, where sensors move towards events as and when they are generated. The objective is to enhance event coverage without compromising existing coverage (i.e., create new coverage holes). In [18], Wang, Cao and La Porta consider a heterogeneous sensor network, where the goal is to enable cooperation between mobile and static sensors to enhance event coverage. In the algorithm, mobile sensors are treated as servers to respond to events. Each mobile sensor has a certain base price for covering an event in the sensing field. The price is related to the size of any new coverage hole generated by its movement.

In [19], a hybrid sensor network consisting of static and mobile sensors is considered, where static sensors are used to detect events, and mobile sensors move closer to such events as and when they are detected. In the algorithm in [19], the problem of moving mobile sensors to events is reduced to that of a maximum-matching problem in a weighted bipartite graph.

In [20], a problem similar to the one in this paper has been addressed. A set of events take place in a network and algorithms are designed for the sensors to cover the events with low delay. However, the location of events and their arrival and departure times are known. The problem we address here is a generalized version of this problem, where we consider the case where the location and distribution times of the events are not known.

4. OUR PROPOSED METHODOLOGY

Consider a convex domain on the plane, with non-empty interior, which we will refer to as the network area or workspace. A stochastic process generates events over time, which are associated to points in Ω. The process generating events is modeled as a spatio-temporal Poisson point process, with temporal intensity $\lambda > 0$, and a spatial distribution described by the density function $\Phi :\to \Re+$.

Given the lack of specific information on event occurrence locations, the best strategy for the mobile sensors is to continuously traverse the entire network area monitoring for events. In this paper, we propose an efficient, and yet simple strategy for traversing the network area. The efficiency is based on minimizing the overall movement distance traversed by sensors, while still maximizing area covered. To this end, we first compute the corresponding contour for the sensors to traverse. Then, we develop a strategy for traversing the path in presence of multiple sensors. We also derive the analytical bounds for the probability of detecting the event occurrences.

4.1. Algorithm Description. First, we construct a contour C such that any point P in the network is either on C or is within a distance of r from C. Thus, this construction ensures that a continuous travel path of C guarantees eventual surveillance of the entire network. Let v be the speed of the mobile sensors and let $|C|$ indicate the length of the contour C. Then, τ, the time required for a sensor to tour C once is given by $\tau = |C|/v$.

In the scenario where the events do not expire until detected and there is only one mobile sensor, the worst case delay is τ and best case delay would be zero (when the event occurs within range of the sensor at time zero). The average detection is $\tau/2$.

Consider the scenario where m mobile sensors are present, each moving (in clockwise direction) around C with velocity v. The distance between two adjacent

sensors is assumed to be the same and equal to $|C|/m$. The case where distance between adjacent mobile sensors is less than $2r$, i.e. $C/m \leq 2r$, is trivial since in this case, we can simply position the sensors and achieve 100% network coverage monitored at any point of time. Thus, all the events would be detected with *detection delay*= 0. Such cases, where achieving coverage and minimizing detection delay are less interesting, are referred to as dense deployments. We will focus our attention on sparse deployments where $|C|/m > 2r$, i.e. $m < |C|/2r$.

For such an algorithm to work, the sensors must either be initially deployed using instructions from a central controller, or the sensors could utilize a protocol that ensures initial placement. In general, however, a uniform random deployment where each sensor begins by assuming it is at the center of its contour could accomplish the same goal, albeit with the possibility of coverage holes or clustering of sensors if some mechanism were not employed to prevent such cases.

4.2. Monitoring the Entire Network (Ideal Case). When number of sensors is sufficient to cover the entire network, then we simply position them in strategic locations so that the entire network is monitored all the times. We can compute the strategic locations using the *Covering Problem* stated as follows: "What is the minimum number of circles required to completely cover a given 2-dimensional space?"

Figure 1 shows three examples. If we divide the coverage area into a grid, where the width/height of each cell is less than $\frac{r}{\sqrt{2}}$ as shown on the right in the figure, then a sensor placed anywhere within a cell can sense any event within that cell. Thus placing at least one sensor in each cell is sufficient to cover the entire coverage area, and a tour need only reach any point within each cell to ensure detection of all events. We can eliminate some redundant coverage by increasing the size of the cells. If the width/height of each cell is less than $\frac{2r}{\sqrt{2}}$ as shown in the center in Figure 1 then a
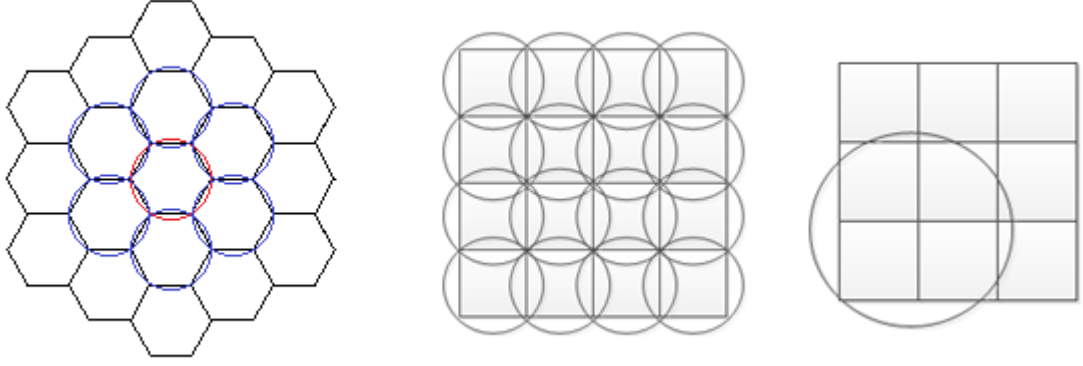
Figure 1. Patterns for overlapping circles for blanket coverage.

sensor located at the center of a cell is able to detect an event anywhere within that cell.

Kershner [21] showed that no arrangement of circles could cover the plane more efficiently than the hexagonal lattice arrangement shown on the left in Figure 1. Initially, the whole space is covered with regular hexagons, whose each side is r and then, circles are drawn to circumscribe them.

In the ideal case, we can have one sensor positioned at each hexagonal center monitoring for events. In such a scenario, since the entire network is being monitored, any event would be detected with probability 1.0 with the detection delay virtually being zero. Thus, when area of the network is large compared to the area of one circle, total number of hexagons n can be approximated as:

$$n = \frac{2A}{3\sqrt{3}r^2} \tag{1}$$

where A is the area of the entire network, and n is produced by dividing the total area by the area of a single regular hexagon with side length equal to the sensor range.

If the area covered by a sensor stays the same relative to the total area being searched (which we will define as the density index, or I_d, then the coverage performance of the network stays the same. So the performance of the mobile sensor network is really dependent on the number of sensors relative to I_d. Consider the three approaches to analysis of sensor coverage performance illustrated in Figure 1. If the algorithms are to cover the same area using equivalent number of homogeneous sensors, we can compare the inefficiency of the algorithms in how they solve for 1-coverage of the search area.

For an algorithm that defines coverage as having sensors cover a square cell from the center of the cell as in Figure 1, far right, we need a cell size of 10x10 and sensor range $r = 10 \cdot \sqrt{2}/2 \approx 7.071$. The coverage area of a sensor is $c = \pi r^2 \approx 157$. Therefore with 100 sensors we cover 15,700 units which amounts to 5700 units of waste, or 157% of the coverage required for 1-coverage of the search area.

For sensors that can cover a square cell from anywhere in the cell, as in Figure 1, center, the waste gets even worse. We would need a cell size of 10x10 and sensor range $r = 10 \cdot \sqrt{2} \approx 14.142$. The coverage area of a sensor is $c = \pi r^2 \approx 628.319$. Therefore with 100 sensors we cover 62,832 units which amounts to 52,832 units of waste, or a whopping 628% of the coverage required to achieve 1-coverage.

For sensors that can cover a hexagonal cell from the center, Figure 1, far left, we would need regular hexagonal cells with side length (equal to the sensor range) sufficient to divide the search area into 100 equal-sized hexagons.

$$\text{Area} = A = 10000$$

$$\text{Area of Hexagon} = A_H = \frac{3\sqrt{3}}{2}r^2$$

$$100 \cdot A_H = A \tag{2}$$

Next, we solve for r^2. From Equation 2 above, we substitute for A_H and A which produces:

$$100 \cdot \frac{3\sqrt{3}}{2} r^2 = 10000$$

$$r \approx 6.204$$

Thus for this approach, the coverage area of a sensor is $c = \pi r^2 \approx 120.92$. One hundred such sensors produce only a mere 2092 units of waste for a very frugal 121% of coverage required to achieve 1-coverage of the search area.

4.3. Construction of Contour for Efficient Monitoring (Practical Case). This case corresponds to more realistic scenarios when the number of mobile sensors is not sufficient to simultaneously cover the network entirely. Moreover they might be able to cover only a small area at any time.

In this scenario, one can simply position the sensors such that the sensors' covering areas would not overlap in the hope that events occur only in the part of the network being monitored. Then, the probability of event detection and detection delay are given by:

$$P_{lower} = \frac{\pi r^2 m}{A}, \; T_{lower} = 0.$$

However, we can trade off some delay for improving the chances of detecting an event. In other words, the sensors can keep moving around in the network to be able to detect more events. This paper addresses the design of an efficient mobility plan for the mobile sensors under the following constraints:

- Maximize the number of events detected by the sensors, before the event expires, and

- Minimize the delay in detecting an event.

We propose to construct the contour C in the following way: Cover the network area with a hexagonal lattice, with the side of each regular hexagon being equal to sensing range (i.e., unity). The contour C would be a simple closed curve constructed as follows:

- Contour C consists of line segments, where each line segment connects some pair of hexagonal centers, and

- Each hexagonal center appears once and only once on the contour C.

Figure 2 illustrates one such construction. The mobile sensors are placed equidistant along the contour. In other words, the path distance between any two immediate sensors is $|C|/m$, where m is the number of mobile sensors. The sensor motion strategy would be continuous traversal of C in counter-clockwise direction (or clockwise direction).
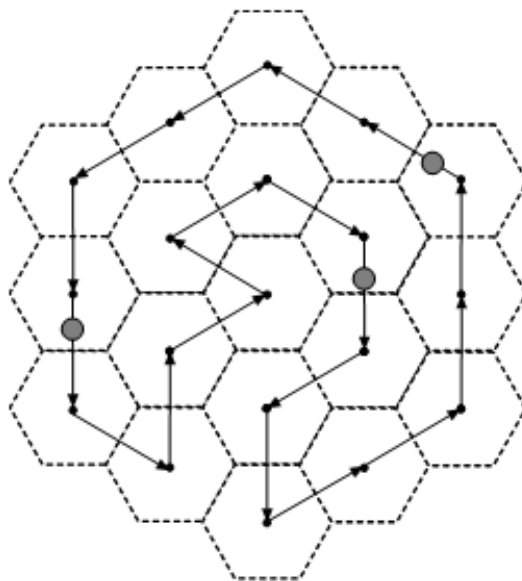
Figure 2. Illustration of construction of C.

```
r <- 2 // number of concentric circles
n <- 0 // current circle
do while n < r
{
  if n = 0 then
    move(northwest) // enter next ring
  else
    move(north) // enter next ring
  n <- n + 1
  if n is even then
    goclockwise(n, r)
  else
    gocounterclockwise(n, r)
  for i <- 1 to r
    move(south) // back to origin
}
goclockwise(a, b)
{
  for i <- 1 to a - 1
    move(southeast)
  for d in {south, southwest, northwest,
    north}
    for i <- 1 to a
      move(d)
  for i <- 1 to a - 1
    move(northeast)
  if a = b then
    move(northeast)
}
gocounterclockwise(a, b)
{
  for i <- 1 to a - 1
    move(southwest)
  for d in {south, southeast, northeast,
    north}
    for i <- 1 to a
      move(d)
  for i <- 1 to a - 1
    move(northwest)
  if a = b then
    move(northwest)
}
```

Figure 3. Algorithm for traversing contour.

4.4. Algorithm for Construction of Contour. Figure 3 describes the means by which a sensor traverses a contour such as that shown in Figure 2.

Note first that the directions specified need not be actual directions, but simply a set of eight angles $\pi/4$ radians apart from one another. The algorithm can support any number of concentric rings. Each ring is entered and then the ring is traversed in the opposite direction from the previous ring, leaving a path northward from the origin for the sensor to return to the origin after traversing the outermost ring, completing the path. Supporting this basic algorithm are two functions goclockwise and gocounterclockwise, also shown in Figure 3.

5. PERFORMANCE ANALYSIS

In this section we will define the length of the contour C then define the performance of the traversal strategy detailed in Section 4.4.

5.1. Length of Contour. Part of the basis for determining the coverage efficiency is determining the length of the contour between hexagons.

Lemma 5.1 (Length of contour)**.** *We note that the length of the contour constructed in the above manner is* $|C| = 2\sqrt{3}rn$, *where* r *is the distance from the center to any of the vertices (and the radius of the circle that circumscribes the hexagon), and* n *is the number of hexagons required to cover the network.*

Proof. Distance between the centers of two neighboring hexagons is $2\sqrt{3}r$. Given that there are n hexagons implies that C is a curve through n points (each point being center of a hexagon). Since, there are n points, we need n segments to connect all of them together, with each segment being a line joining two neighboring centers and hence of length $2\sqrt{3}r$. Thus, the total length $|C|$ formed with n line segments, each of length $2\sqrt{3}r$ is $2\sqrt{3}rn$. $\square$

From Lemma 5.1, the length of a contour C constructed using a hexagonal lattice is $2\sqrt{3}rn$. From Equation 1 and Lemma 5.1, and by setting $r = 1$, the total area covered by a sensor traversing the contour C once completely can be approximated to $4A/3$, noting that $A \gg \pi$.

5.2. Calculating Coverage Efficiency from Movement Strategy. Coverage efficiency is a metric we define in order to express the area being actually covered by the sensors compared to the maximum area they must ideally cover cover. In other words, coverage efficiency quantifies overlaps in the coverage during sensor movements in our algorithm. Here we establish a milestone, then compare our strategy in those terms.
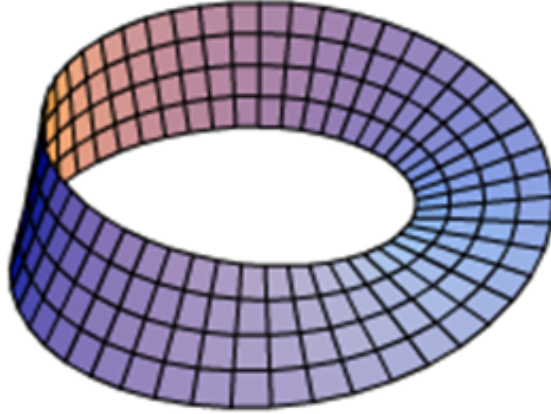
Figure 4. Illustration of Möbius strip.

One sensor with range r traversing a Möbius strip (as shown in Figure 4) of width r and length d at velocity v units per second will have probability of detecting an event of duration u seconds as computed by:

$$P = \frac{d \cdot u}{v} - \pi r^2$$

It will take d/v seconds for the sensor to cover the entire area. Any event with duration $u \geq d/v$ has $P = 1.0$. In addition, the coverage area is dr, and the sensor covers $2\pi r^2 v$ per second.
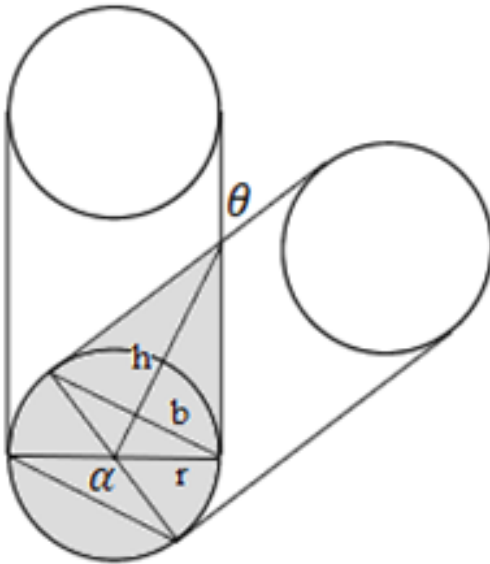
Figure 5. Geometric description of overlapped sensor coverage area.

When a sensor with sensor range r is traveling straight ahead for distance d, the coverage area is $\pi r^2 + 2rd$. When the sensor travels for distance d, then makes a turn and travels another distance d (as shown in Figure 5), then as long as the distance between the start position and end position is greater than $2r$, the total coverage area can be computed by doubling the previous value, then subtracting the area of overlap. Thus, the coverage area A is:

$$A = 2(\pi r^2 + 2rd) - \pi r^2 + t - l$$

Where t is the area of the triangle computed as bh and l is the area of the lens within the triangle. The area of overlap $O = \pi r^2 - t + l$. Note for the following formulae

that $\theta = \pi - \alpha$.

$$b = r\sin\alpha$$

$$h = \frac{b}{\tan\theta} = \frac{r\sin\alpha}{\tan\theta} = \frac{r\sin\alpha}{\tan(\pi-\alpha)} = \frac{r\sin\alpha}{-\tan\alpha}$$

$$t = bh = -\frac{r^2(\sin\alpha)^2}{\tan\alpha}$$

$$l = \frac{1}{2}r^2(\alpha - \sin\alpha)$$

Using the formulae above, we can calculate the coverage area A as:

$$A = 2(\pi r^2 + 2rd) - \pi r^2 + \frac{r^2(\sin\alpha)^2}{\tan\alpha} + \frac{1}{2}r^2(\alpha - \sin\alpha)$$

If $\alpha = 3\pi/4$, the sensor range $r = 1$, and the distance $d = 2\sqrt{3}$ the coverage area would be:

$$A = 2(\pi + 4\sqrt{3}) - \pi + \frac{(\sin 3\pi/4)^2}{\tan 3\pi/4} + \frac{1}{2}(3\pi/4 - \sin 3\pi/4)$$

If the number of rings $n = 0$ (a single hexagon) then the coverage area is πr^2. If $n = 1$, then there are 7 hexagon centers visited, four $\pi/4$-radian turns and three $3\pi/4$-radian turns. If $n > 1$, then the number of hexagons h will be:

$$h = 1 + \sum_{i=1}^{n} 6i$$

In such a configuration, the number of $3\pi/4$-radian turns is $n + 2$, and the number of $\pi/4$-radian turns is $6n - 4$.

Thus for any number of rings in our network, we can determine the coverage area, and from that the coverage efficiency.

Admittedly, the coverage distribution is not uniform when the sensing device has coverage that is circular in shape. As the sensor moves straight ahead, events

located along the center of the path will be detected with greater probability than events near the edges of the coverage *swath.*

Lemma 5.2 (Probability of detection of an event). *Consider an event i with lifetime t_i. The probability that the event is detected before it expires in a sensing field with total area A_T given m mobile sensors traversing at a velocity v is given by:*

$$P_{(m,t_i)} = \min(1, \frac{m(\pi + d_k)}{A_T})$$

Proof. The distance traversed by a mobile sensor k in time duration of t_i is $d_k = v * t_i$. If $|A|/m \leq (2r + d_k)$, $(r = 1)$, then before the event expires, the sensors are able to collectively traverse the entire contour, thus sensing the entire network area. Thus, the probability of detection is unity.

If $|A|/m > (2r + d_k)$ with $(r = 1)$, then the total area sensed by the sensor k can be computed as follows:

$$A_k = \frac{\pi(1)^2}{2} + 1 * d_k + \frac{\pi(1)^2}{2} = \pi + d_k$$

The collective area covered my m sensors is then $m * A_k$. The event would be detected if the event is located within this area monitored. Thus detection probability is $m * A_k / A_T$.

Combining both the above cases,

$$P_{(m,t_i)} = \min(1, \frac{m(\pi + d_k)}{A_T})$$

□

Lemma 5.3 (Maximum Detection Delay). *Consider an event i with lifetime t_i. Maximum detection delay D for the event is:*

$$D = \min(t_i, \frac{(|A|/m) - r}{v})$$

Proof. For D to be maximum, the event's location has to be the farthest point from current location of any sensor in the direction of path traversal and not in the sensing coverage of any sensor. For instance, if C_1 and C_2 are the current locations of sensors, then l_i is one such location: it is not being sensed by C_2 and the sensor at C_1 has to traverse a distance of $(d - r)$ before the event can be sensed, where $d = |A|/m$. The corresponding detection delay is $(d-r)/v$, where v is the velocity of the sensors. However, if $t_i < (d - r)/v$, then the event expires before the sensor can traverse the corresponding distance. In such scenarios, the maximum delay corresponds to events that require sensor traversal of a distance of $v * t_i$. Putting the above two scenarios together, the maximum detection delay is

$$D = \min(t_i, \frac{(|A|/m) - r}{v})$$

□

6. PERFORMANCE EVALUATION

We developed a simulator that first constructs a hexagonal tiling for a given network area and then computes a contour C for the traversal of sensors through the centers of the hexagons. Once the contour is computed, the sensors are placed equidistant on the contour. The sensors continuously traverse the contour for the entire duration of simulation. Events are randomly generated over the network area. We consider two kinds of events: never-expiring events and those whose lifetime
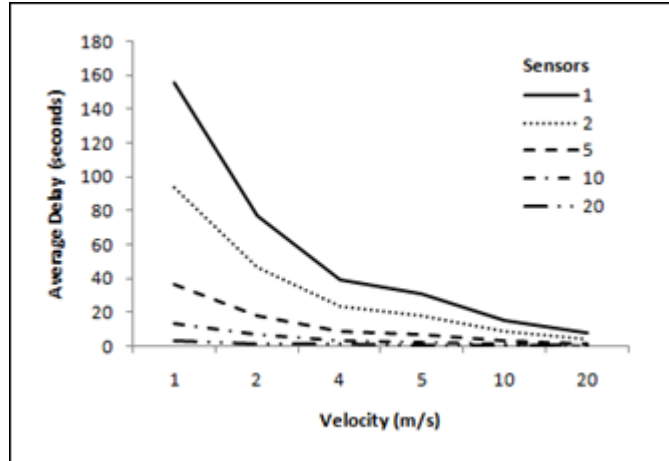
Figure 6. Average Detection delay for varying sensor speeds for the scenario where the events do not expire until detected.

follows a Poisson distribution with a given mean. We use never expiring events to compute maximum and average detection delays for varying number of mobile sensors. For events with limited lifetime, the probability of event detection is the metric we study.

To minimize the edge effects of a network, we chose a hexagonal network area with a side length of 60m. The sensing range is 10m. Thus, the hexagonal tiling consists of 19 hexagons (we ignore small gaps near the network edges). Figure 6 shows the protocol performance for never expiring events for varying sensor speeds and shows the effect of varying sensor velocities on the detection delay. As the velocity of the mobile sensors increases, the delay decreases. Naturally, enough sensors could be added so that the entire tour could be covered continuously.

Here we would like to note the difference between doubling the speed of the sensors versus doubling the number of sensors: doubling the speed results in a greater improvement in the detection delay. With velocity fixed at 1 m/s, the detection delay with 5 sensors is 30.8955 compared to the case when the number of sensors is fixed at 1, the detection delay at velocity 5 m/s is 36.7074 seconds. Increase in number of
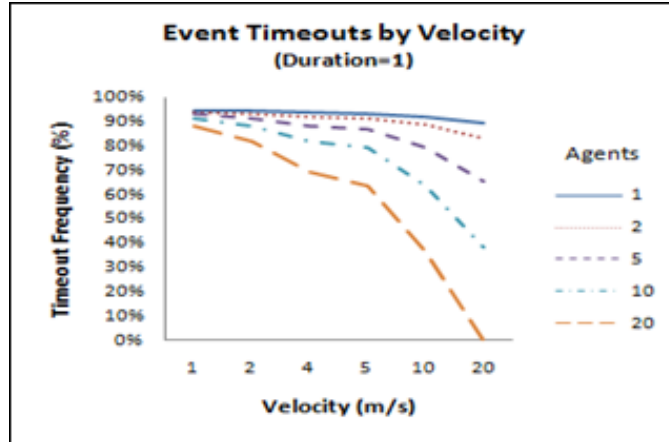
Figure 7. Timeout Frequency vs. Velocity with Duration=1.

sensors reduces the delay more than increasing the speed. To understand the reason behind this, we remind the reader additional area coverage is computed as a function of number of sensors and speed of the sensors. We note that any change in number of sensors (m) impacts two terms, while the speed (v) impacts only one term.

In Figures 7, 8 and 9, we plot the percentage of events that are un-detected in the network with respect to the sensor velocity for different event durations and number of sensors. We find that as the velocity increases the percentage of undetected events decreases. Furthermore, when the event duration and the number of sensors increases, the percentage of undetected events also goes down dramatically.

7. FUTURE WORK

Additional exploration may include exploring the applicability of the ring pattern contour to certain sensor field shapes. A contour where the sensor makes $\pi/2$-radian turns could produce similar results and be more applicable to more rectangular-shaped sensor fields, whereas the applicability of the hexagonal rings is shown here to provide efficient coverage considering only circular sensor fields.
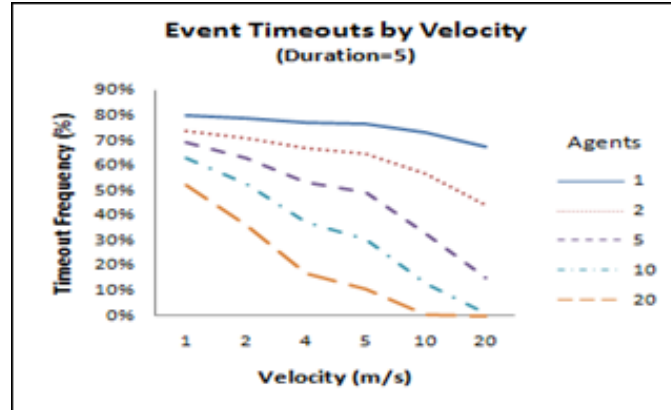
Figure 8. Timeout Frequency vs. Velocity with Duration=5.
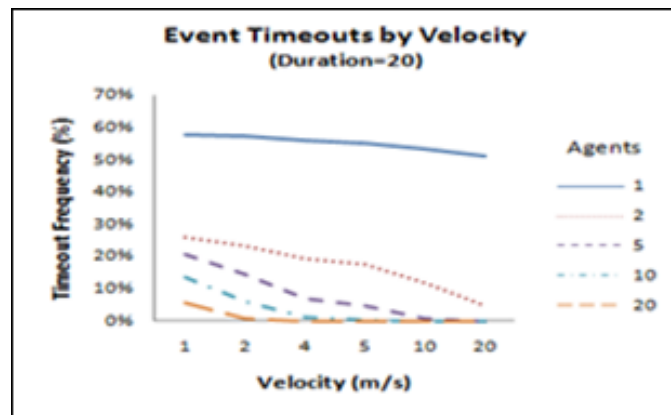
Figure 9. Timeout Frequency vs. Velocity with Duration=20.

Other variables besides velocity can also be considered, such as adjusting the sensing frequency to conserve power, adjusting sensor range, etc.

8. CONCLUSIONS

In this paper, we considered a representative surveillance problem wherein a sparse number of mobile sensors are expected to cover events of interest in a deployment field. The location and duration of each event in the network is unknown. To address this problem, we designed a movement algorithm based on efficient contour

traversal for the sensors in the network for two objectives: maximizing event detection, while minimizing detection delay. Using extensive analysis and simulations, we demonstrated the performance of our algorithm from the perspective of event detection and delay with respect to the number of sensors, movement velocity, and the number and duration of events.

PAPER II. EXPLORATORY COVERAGE IN LIMITED MOBILITY SENSOR NETWORKS

1. ABSTRACT

The impact of limited mobility in mobile sensor platforms is a limiting factor in the effectiveness of coverage algorithms. As the usage of autonomous drones increases in applications ranging from search-and-rescue, detection of forest fires, and surveillance, the ability for these platforms to employ distributed algorithms is increasingly important. In this paper, we examine the problem of Exploratory Coverage, wherein the objective is to move a number of mobile sensors to fully explore (and hence, sense every point in) a target area in order to detect any critical event that has already occurred in the area. Further, we present a taxonomy of coverage problems as identified by the relationships between sensor range, coverage area, number of sensors, and mobility (range). We then design a purely localized and distributed approximation algorithm for our problem, and provide simulation results for sparse and balanced deployments to demonstrate the effects of limited mobility on Exploratory Coverage.

2. INTRODUCTION

In this paper, we study the problem of Exploratory Coverage. The notion of Exploratory Coverage is different from traditional notions of coverage such as Blanket, Sweep, or Barrier Coverage [9]. Blanket Coverage involves deploying sensors such that every point in the coverage field (the area to be searched) is covered by one or more sensors, as if waiting for future events to occur. We express Blanket Coverage as $\forall (x,y) \in A, \exists s \in K : |(x,y), s| < r$, where A is the search field, K is the set of mobile sensors deployed within A, assuming that centered on the location of each mobile

sensor is a uniform disc within which the probability of detection for any event within distance r of that center is 1.0. Sweep Coverage has been described as a moving wave of sensors and is sometimes solved by periodically polling points-of-interest that are known a priori. And in the latter, Barrier Coverage, every crossing path between the boundaries of the coverage field must be covered by one or more sensors, as if sensors lie in wait for a mobile intruder to pass by. Mobility (and limited mobility) has been studied in these types of coverage problems as well. These various notions fall under the class of proactive coverage, wherein sensors are proactively deployed to sense events of interest. Exploratory Coverage focuses on sparse deployment models, wherein ensuring full blanket coverage of the field would require too many sensors to be deployed. Thus, we leverage mobility (albeit limited) in sensors to enhance quality of exploratory coverage with fewer sensors. In Exploratory Coverage, the strategy is to move a number of mobile sensors from an initial deployment in an attempt to eventually fully explore (and hence sense, or cover) every point in the coverage field at least once in order to detect any critical event that has already occurred in the area.

Our work utilizes limited mobility sensor platforms such as those mentioned above by conducting an initial deployment, then employing an algorithm to direct the mobility of the sensors in order to locate events in the coverage field. With a solution to Exploratory Coverage, the mobile sensors can make the best use of their limited mobility in order to reach as much of the coverage field as possible. We study the key parameters: number of sensors, sensor range, and size of coverage field. Together these parameters determine coverage density and partition the problem set into subproblems. Together with the parameters of mobility range and deployment concentration, we are able to study the effects of limited mobility on area coverage.

To determine the effectiveness of the solutions examined, we measure the area that is covered by the sensors from their initial deployment until mobility stops, either

because mobility range has been reached by all sensors or the algorithm employed fails to result in any sensors choosing to move. We also consider the amount of new coverage over time (the rate of new coverage) and additional coverage added as return on investment for additional mobility range.

There are numerous factors driving increased attention to usage of automated drones. First, human safety factors are a primary motivator for interest in employing robots for a variety of tasks where human presence is not preferred. Using drones in battlefield scenarios remove humans from dangerous situations. Using drones for search and rescue in situations where hazardous materials are involved reduces the need to risk additional human exposure. Second, even in applications where human safety is not an issue, using unmanned platforms often has a significant economic advantage. Mobile sensor hardware platforms have seen many recent advances such as higher computation power, relatively lower weight, and lower power requirements that allow drones to carry much more computational capacity and payload or stay deployed and functional for longer periods of time. More and more, these platforms are readily available off the shelf, as the benefits of mass production of commercially designed systems are realized.

Research into advances for robotic platforms is occurring on many fronts [1]. UAV Platforms such as the DraganFlyer$^{\text{TM}}$X6 [2] as seen in Figure 1.1 show great promise as both an academic as well as practical model in real-world commercial applications. Less recently, numerous efforts have focused on building self powered miniature mobile robot sensor platforms as shown in Figure 1.1 on Page 3. The Robo-mote (70mm x 45mm x 35mm) [3], Khepera series of robots (70mm x 30mm) [4] [5], and the XYZ platform [6] are prototypes of mobile sensors that are battery powered and motor driven. In each of these platforms, sensors (to sense events) and commu-nication devices create mobile communicating sensors. A rather novel approach to

mobility in sensor platforms utilizes a fuel-powered hopping mechanism [7] used in Intelligent Mobile Land Mine Units (IMLM) developed by DARPA.

We first review related work in this area, then formally define our problem including a taxonomy of the problem space and the subset to which we will devote our focus. Given the complexity of the problem, we analyze two centralized and one distributed approximation algorithms. One of the centralized algorithms constrains the number of mobility choices ("hops") that a sensor can make, where the distance traveled per choice is considered to be constant, and the other places the constraint on the distance traveled by each sensor. The distributed algorithm takes the constrained choices approach, but each sensor uses a localized heuristic to make mobility decisions. Finally, we analyze the performance of the distributed solution, fixing certain variables and varying others to understand their effects on coverage. Lastly, we present our conclusions about the effects of mobility on this type of coverage problem.

3. RELATED WORK

Our problem has a number of close relatives in the class of Traveling Salesman Problems (TSP) [22], where an agent, given a graph representing cities at the vertices, and distances between cities as the edges, is tasked with visiting all the cities while traveling the shortest distance.

A variant known as the Prize-Collecting Traveling Salesman Problem (PC-TSP) [23], charges the agent a cost for traversing each edge, and pays the agent a reward upon arrival to each city, while loosing the requirement that the agent visit all cities. The goal is to maximize the overall reward, which is particularly relevant to our problem since we are measuring algorithm effectiveness relative to mobile sensors reaching new, as-yet-uncovered areas. Another variation of PC-TSP that includes reducing rewards over time is known as the Discount-Reward TSP (DR-TSP) [24].

Modeling an agent-controlled sensor as a salesman, the coverage field as a graph, and assigning costs/rewards for touring the graph and visiting nodes, where rewards diminish over time and by how many sensors have already visited a node makes these techniques closely analogous.

In a similar fashion, we can say our problem is also closely related to k-TSP [25] and Vehicle Routing Problem (VRP) [26]. In k-TSP, k agents can start at various cities and must solve for k tours such that the total distance traveled between cities is minimized. A key difference between this problem and ours is that in k-TSP, no single agent has any bounds on its travel distance as long as the overall distance is minimized, where our agents each have limited mobility, or a per-sensor budget, if you will. Often k-TSP and VRP variations involve multiple agents originating at a common (or set of common) point(s). However, we believe that our problem shows promise for numerous real-world example applications where the initial distribution is uncontrolled. Limited mobility is analogous to TSP subtour elimination constraints in the k-TSP problem [27] and capacitation or time-window constraints in VRP [28].

One differentiating factor between the problems is the construction of the coverage field. While analogous, TSP and VRP problems typically apply to problem sets involving graphs representing roads, whereas our problem applies to a continuous coverage field (conceding that we approach the problem with a discretized implementation by dividing the area into tiles that can be considered a graph where each node is connected to nodes representing adjacent tiles). The characteristics of the graphs could be seen to vary only in terms of degree of connectivity.

Another close relative is the problem of ORIENTEERING [29]. In the Orienteering problem, the challenge is to find a walk between two nodes that maximizes the total number of nodes visited within a travel budget. In our problem, in contrast, the goal is to maximize coverage, but there is no destination (referred to as the

unrooted variant of the problem), simply that as much of the coverage field is visited as possible before mobility limits are reached.

Blanket coverage has been approached using other methods, such as potential fields [30], where navigation is directed based on a model of repulsive forces and friction to encourage equilibrium. Another approach used in dense deployment configurations is to identify as many disjoint sets of sensors as possible that cover the area completely and activate them successively [31]. Other approaches, such as ant colony optimization technique, have also been employed [32].

Barrier and sweep coverage have been studied using a TSP approach [33] [34], as well as by other techniques, including a technique where the coverage field is decomposed into subregions [35]. Other approaches, such as machine learning techniques, have also been employed [36].

4. PROBLEM DEFINITION

We formally define the problem of Exploratory Coverage as follows. First, we are given a planar area defined as the coverage field A. Next, we are given a set of mobile sensors K each with sensor range r and mobility range d. For the distributed algorithm, we further assume sensor capabilities to include a communication range that provides awareness of neighboring sensors. Sensor range r specifies the radius of a disc within which a sensor can detect an event, and mobility range d is the total distance that a mobile sensor can travel. Mobility range is sometimes referred to as the overall distance that can be traveled, and sometimes as the number of units of distance traveled for each mobility choice, without loss of generality.

Further, we establish an initial deployment function $f(K, A)$ that produces an initial deployment location for each sensor in K. As we will see later, we focus on a

function that uses a Gaussian distribution around a point for this function, such as might occur when sensors are dropped from an aircraft.

We define a navigation to be a set of transitions (using mobility) of a sensor from an initial deployment location to a final resting location, when either that sensor's mobility range would be exceeded by another move, or the sensor has decided to stop navigating (say because A has been covered and additional mobility would only waste resources). From a given initial deployment configuration, we must produce a set of navigations (one each for sensors in a set) such that the execution of those navigations results in maximal coverage of A. The number of all possible navigations from all possible starting locations to all ending locations can be very large. The subset that would actually result in complete coverage of A can also be large. On the other hand, many potential navigations that would result in complete coverage could be inviable if they require exceeding the mobility constraints of one or more sensors.

Next, we define a plan L to be a set of navigations. Consider the set of all plans to be $C = \{L_1, L_2, ...\}$, and D is the subset of plans that result in complete coverage of A without exceeding mobility constraints for K. We seek to map an initial deployment to a plan such that we maximize the area covered by the sensors while minimizing the total distance traveled by that set of sensors. In other words, we seek to select a set of mobility choices for sensors such that navigating according to this plan results in maximizing event detection in A while minimizing the number of moves.

The decision problem can be expressed as: given C, A, K, r, d, and f, is there a plan $L \in C$ that would result in covering all points in A? We note that there exist instances of the problem for which there are no such plans, and so we also discuss the problem of finding a plan that maximizes coverage of A while minimizing total distance traveled by sensors.

5. TAXONOMY OF SUB-PROBLEMS

There are numerous ways to attempt to produce an optimal number of sensors to cover a region. In a deployment where the number of sensors relative to the size of the search area is very small, the maximum coverage is the number of sensors times the area of the coverage disc defined by the sensor's range, or $|K| \cdot \pi r^2$. When we want to know the minimum number of sensors required to achieve blanket coverage, then we must acknowledge the overlap required to cover all points. In Figure 1 we see three ways to define coverage commonly found in literature. We can arrange sensors so that they circumscribe (inscribe) the circles about the hexagons of a regular hexagon network, which has been shown to be optimal [37]. Another common way that sensor coverage is analyzed is to divide the coverage field into a grid of squares such that sensors at the center can sense all points in a tile. This creates a larger overlap than using hexagons, but can be useful when the problem at hand involves touring the centers of the squares. Defining a grid such that a sensor at any point within a tile can sense every point in that tile creates more overlap still. For this discussion, we will choose the optimal configuration using inscribed hexagons.

Coverage problem instances can be partitioned into a number of sub-problems, where each sub-problem is identified by the relationship between the number of sensors, the ratio of the sum of the potential coverage area of all sensors combined to the size of the coverage field, and the mobility constraints of the sensors. We partition these problems according to the definition of $n \approx \frac{2A}{3\sqrt{3}r^2}$ using the pattern of inscribed hexagons from Figure 1.

Partition 1: "Sparse", i.e., $|K| < n$. There are problem instances for which there is no solution whereby the mobility of the sensors will allow for even 1-coverage to be achieved. In these cases, the best we can hope to achieve is a solution that improves coverage from the initial deployment to covering as much of the coverage

field as possible given the available mobility of the sensors. In this case, we consider feasible solutions to be those that result in maximal coverage.

Partition 2: "Balanced", i.e., $|K| \approx n$. There are instances where a plan exists (there exists at least one $L \in D$) where all of A can be covered within mobility constraints.

Partition 3: "Dense", i.e., $|K| > n$ or $\frac{r^2}{A} > \frac{4|K|\sqrt{3}}{3}$, $d = \epsilon$. There are instances where the number of sensors is high enough, and settings for r and constant d provide for the sensors to achieve n-coverage, where we are able to show that plans exist such that multiple sensors can pass within r of every point in A (or even achieve blanket coverage of A). Sometimes this is a beneficial problem space to explore, e.g., in cases where the coverage field need be covered for a time that can exceed the lifetime of the sensor. In this case, the goal is to provide redundant coverage so that as sensors fail, 1-coverage can be achieved multiple times (or preserved for a maximal period of time).

Partition 4: "Increasing mobility", i.e., $d \to \infty$. There are instances where the settings for r and d can provably be shown to allow most plans in C to feasibly cover A, regardless of the initial deployment ($D = C$). We resist stating that all plans can cover A since it is easy to construct plans that can't, i.e., plans where sensors do not try to move toward previously uncovered points in A.

A visualization of this simple taxonomy of coverage problem instances is shown in Figure 1.

For this paper, we focus on partitions 1 and 2 which we categorize as Sparse Coverage problems. In these problem instances, plans may exist to allow coverage of A but are not guaranteed that such plans exist. Rather, we are dependent upon the initial deployment and a thoughtful mobility strategy to achieve success.

In partition 3 where the number of agents or the sensor range of those agents becomes such that much of the coverage field can be covered by multiple sensors
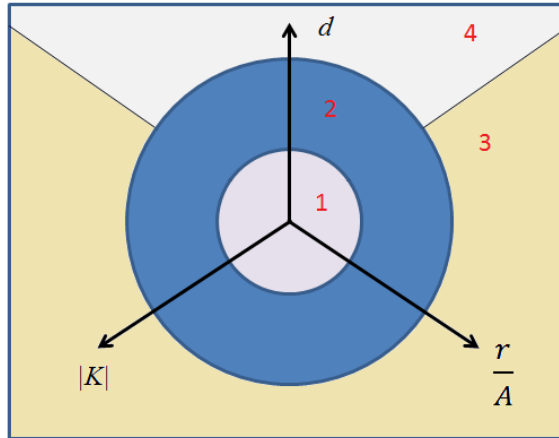
Figure 1. Taxonomy of Coverage Problem Instances

simultaneously, then the problem shifts closer to the realm of other coverage problems such as Blanket coverage, and interesting solutions to explore focus on n-coverage rather than 1-coverage.

In Figure 1, a solution to partition 4 cases can be constructed in a straightforward manner. When the mobility budget is no longer an issue, then the problem becomes k-TSP. First, we pick a set V of $|K|$ points in A where each point in A is within r of at least one of the points in V. Each sensor is assigned a point from V and navigates from its origin to that point. Thus if not for time, mobility, nor other factors such as hazards and detection, we would simply loosen mobility constraints and wait until each sensor navigated to its assigned post. Since we are considering cases where the budget for mobility is a real constraint, however, more care must be taken in choosing the tour for each sensor to maximize coverage while minimizing distance traveled.

6. APPROXIMATION ALGORITHMS

Approximation algorithms have been developed for many variations of the problems discussed above. There is no known polynomial time solution for general

TSP unless $P = NP$, however the $(1+\epsilon)$-approximation for Euclidean-TSP in $\Re^2$ and $\epsilon > 0$ is relevant to the discussion [38] and as well the 3/2-approximation algorithm for TSP where the triangle inequality holds [39] are well known. However, these algorithms are not practical for our problem as they rely on the location of events to be known a priori. Bounds on approximation schemes for PC-TSP are explored in [40]. For the problem of ORIENTEERING, a 2-approximation algorithm exists when the graph representing the coverage field is undirected [29].

Here we present three algorithms, two centralized and one distributed, for an approximate solution to the Exploratory Coverage problem. For solutions involving constrained hops, we refer to a "hop" as a mobility choice that results in a sensor choosing to move a constant distance from its current location in a chosen direction to a new location where it waits to make another mobility choice.

6.1. Centralized Algorithm - Constrained Hops. The centralized algorithm assumes a centralized processor evaluates the initial deployment and selects a plan for all sensors. First, we choose a target configurations from C such that the points in L_x form a grid of squares of width/height $2^{-1/2}r$. This size is chosen so that any mobile sensor located within this square can sense any point within and thereby covers that square.

From this set of squares, we construct a graph $G = (V, E)$ where $V = v_0, v_1, ...$ is a set of vertices created for each corner of a square, and E is a set of edges created by connecting each vertex in $v_i \in V$ to adjacent vertices to the left, right, above, and below v_i. Each edge cost is thus $2^{-1/2}r$.

In this way, we have constructed a graph where sensors may navigate to cover any portion of the graph by way of hopping from one vertex to another, traveling a constant distance with each hop, and completing their route when they cannot make another hop without exceeding its mobility budget. Thus, we have translated the

problem into one where the number of hops by each sensor is constrained to $d/2^{-1/2}r$ hops.

6.2. Centralized Algorithm - Constrained Distance. For the constrained distance problem, we can first solve the constrained hops problem and then solve ORIENTEERING to go from the constrained hops solution to the constrained distance solution.

The ORIENTEERING problem is the following: given a set of nodes and distances between each pair of nodes, and a budget K, find a tour with maximum number of hops such that the total distance covered is at most K.

With the 2-approximation algorithm for undirected graphs in hand, and a $O(\log^2 n)$-approximation for directed graphs, we can construct a possible solution for the constrained distance problem:

1. Given a constrained distance problem instance X_d, let OPT_d be value of the optimal solution

2. Let X_h be the corresponding constrained hop problem and OPT_h the optimal solution of X_h

3. Solve X_h optimally using the MAX FLOW formalization

4. For each sensor i, let $H(i)$ be the (optimal) set of hops found by the MAX FLOW solution

5. For each i, define an ORIENTEERING problem as follows: $ORIENT_i =$ Given sensor node i's original position and distances to hops in $H(i)$, find a graph with the vertices and distances equal to the shortest path distances. Now this is an instance of the ORIENTEERING problem where the goal is to cover as many vertices as possible using at most distance h

6. For each i, solve $ORIENT_i$ using the approximation algorithm [29]. For each $ORIENT_i$, the solution A_i is a path (of length at most h) over $\{i\} \cup H(i)$ Let OPT_i be the optimal solution for $ORIENT_i$

7. Output $A_1, A_2, ..., A_n$

To argue that this is indeed a good approximation, we need the following:

$$OPT_d <= OPT_h$$
$$<= \sum_i OPT_i$$
$$<= \sum_i 2 * A_i$$
$$<= 2 * value of the solution$$

All these inequalities hold, except the last one. The reason is that the A_i instances could overlap (in terms of tiles they cover), so the value of the solution could be significantly less than $\sum_i A_i$.

6.3. Distributed Algorithm. We can use a MAX FLOW-like solution for the distributed algorithm also. This would be considered distributed MAX FLOW and examples of such a solution can be found in [41]. We can also use heuristic-based distributed algorithms with synchronization. A number of reinforcement-learning techniques such as EA or learning classifiers, and even ant colony optimization approaches would also work.

Another approach to solving the distributed approximation scheme is to employ a heuristic and a form of matching to direct the navigation of the mobile sensors:

1. Given a constrained hops problem instance X_d, let S_0 be the initial state having deployed the mobile sensors

2. For each sensor s_i, use a heuristic to map sensors within s_i's tile to moves that are predicted to produce the highest reward

3. Employ a protocol for s_i and other sensors that occupy the same tile to decide which sensor will perform which move

4. All sensors make a hop

5. Repeat until hop distance would exceed mobility constraint

Next, we discuss the heuristic that the sensors will use to direct their decision making process. The heuristic produces tile colorings defined as:

- White = the tile appears to never have been occupied, and no sensor is in a position where it could move there next turn

- Gray = the tile appears to never have been occupied, and one or more sensors in adjacent tiles could occupy it next turn (we make a tally)

- Black, unoccupied = we remember the tile has been visited, but isn't occupied now

- Black, occupied = it has been visited, and in fact is occupied now (we make a tally)

Additionally, each sensor is mobility-limited and has a remaining number of hops counter.

Once all tiles within each sensor's range is examined and colored, we decide moves based on a simple algorithm.

The basic pseudocode for the distributed solution is shown in Figure 2, which can be stated in two ways: first, how it is implemented, and second, from the perspective of an agent working in distributed fashion.

```
 1: A = {AllAgents}†
 2: while |A| > 0 do
 3:     x = A[0]                                        ▷ select first agent
 4:     if |x.white| > 0 then
 5:         z = random(x.white)
 6:     else if |x.gray| > 0 then
 7:         z = random(x.gray)
 8:     else if |x.emptyBlack| > 0 then
 9:         z = random(x.emptyBlack)
10:     else if |x.lessOccupied| > 0 then
11:         z = random(x.lessOccupied)
12:     end if
13:     if z ≠ ∅ then                                   ▷ move x
14:         x.moveTo(z)
15:     end if
16:     for each agent y in x.nearbyAgents do
17:         y.recalculateWGB
18:     end for
19:     A.remove(x)
20: end while
```

Figure 2. Distributed Solution to WGB †Agents in A sorted by num-white, num-gray, num-empty-black, then by distance descending.

How it is implemented: A turn (or tick) is defined as an opportunity for each agent to move, if it is able to and decides to move.

How it works from the sensor's point-of-view: The main difference is that we simply state that agents employ a protocol whereby they communicate in order to decide who should move first in a neighborhood, and that once that sensor makes a move, it notifies the others and now they recalculate what they know and decide again amongst themselves whose turn it is. The logic is based around the premise that the fewer choices a sensor has, the earlier it moves.

7. PERFORMANCE EVALUATIONS

For a set of sensors to perform optimally, the sensors deployed to a coverage field would navigate the least distance to cover maximal space in the minimum time possible. Because sensor deployments can result in concentrations of sensors to the same area, and with mobility being limited, achieving the specified goal could require more sensors. Alternatively, more hops per sensor could be required in order to assure the desired expected level of coverage. Parameter selection becomes key in producing valuable simulation results. We choose a ratio of sensor quantity to coverage area that produces scenarios characterized as sparse deployments, however we vary the size of the coverage area to allow for observations about mobility decisions to become apparent. We select a variety of values for number of hops and the value of σ in order to evaluate the coverage achieved both following deployment and after the sensors have exhausted their available mobility.

As a side note, in selecting a communication protocol between sensors, the number of messages transmitted is also a factor we desire to minimize, as sending messages costs energy and increases risk of detection (as well as betraying the sensors' location to enemies). Also, although sensor technology options are increasing [42], real-world sensors typically are subject to processing and memory limitations, and so minimizing the utilization of these resources is also important to consider.

In order to gather data to support our hypotheses, we ran a significant number of simulations using combinations of the parameter values shown in Table 1. Input parameters included the width and height of the coverage field (as measured in tiles), the count of sensors, the number of times that a sensor is allowed to hop (its mobility constraint), and a value for σ that is used for the Gaussian around a point deployment type to specify the density of the distribution of sensor distance from the central deployment point.

Table 1. Values selected for simulation runs

Parameter	Value(s)
Hops	1, ..., 8
Concentration (σ)	$\frac{1}{2}$, ..., $\frac{1}{32}$
Area	10x10, ..., 800x800
Sensors	100, ..., 1000

We implemented several heuristics for comparison purposes. In the first heuristic, each sensor stochastically chooses a neighboring tile to which to navigate. As expected, this performed poorly with respect to mobility conservation, given that each sensor always chose to navigate to an adjacent tile. The second heuristic also allowed sensors to navigate to a random adjacent tile, but only if that tile was not occupied by another sensor. This also performed poorer than desired with respect to conserving mobility, but did rather well with respect to coverage. We used these heuristics to measure the relative performance of our White-Gray-Black (WGB) heuristic, constructed according to the distributed algorithm described previously.

The program begins by assigning each sensor to a tile (its initial deployment). We experimented with initial deployment methods, including stochastic and statistical distributions. Results from stochastically assigning sensors to tiles did not produce interesting results, due to the fact that regardless whether we varied the size of the coverage field, or the number of sensors, the only parameter that was changing significantly was the density of the sensor deployment. We experimented with Gaussian distribution of sensors around a point and Gaussian distribution along a line, with satisfactory results. However, the results were similar enough between the two types that we chose Gaussian around a point for the results presented here. Thus, given a point (typically the center of the search area) and a specified value for σ, sensors are more densely located nearer the point, and more sparsely as distance increases
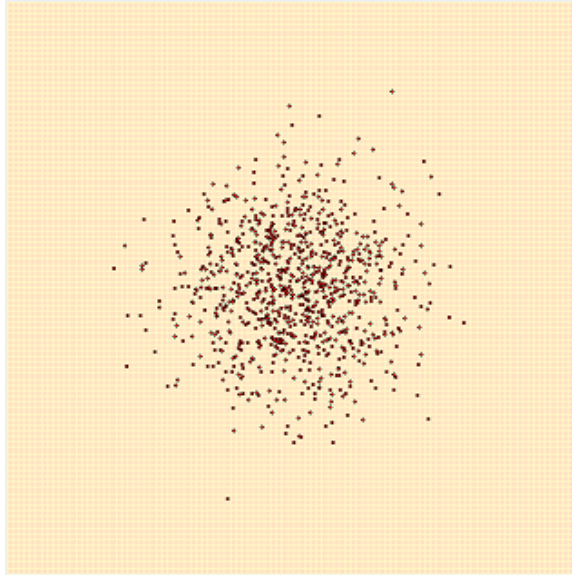
Figure 3. Visualization of Gaussian deployment

from the point. One standard deviation is defined as $stddev = w * \sigma/2$ where w is the width/height of the search area. Thus approximately 70% of sensors should be located within $stddev$ from the center point.

A visualization of a typical initial deployment, where the width/height of the search area is 100, there are 1000 sensors deployed, and $\sigma = 1/10$, is shown in Figure 3. As the simulation runs and the sensors begin exercising their option to hop, they begin to spread out across the search area. A visualization of sensors utilizing their mobility using the WGB heuristic is shown in Figure 4.

The tiles are constructed as a grid with no edges, where sensors that travel North from the top wrap around to the South side, and vice-versa. Likewise, sensors that travel West from the western edge wrap around to the East, and vice-versa.

The simulation begins, and for each trial, agents select navigation choices for the sensors according to WGB until a turn occurs where no moves are made or all sensors hit their mobility limits. Once this happens, the trial ends and this repeats through the specified number of trials.
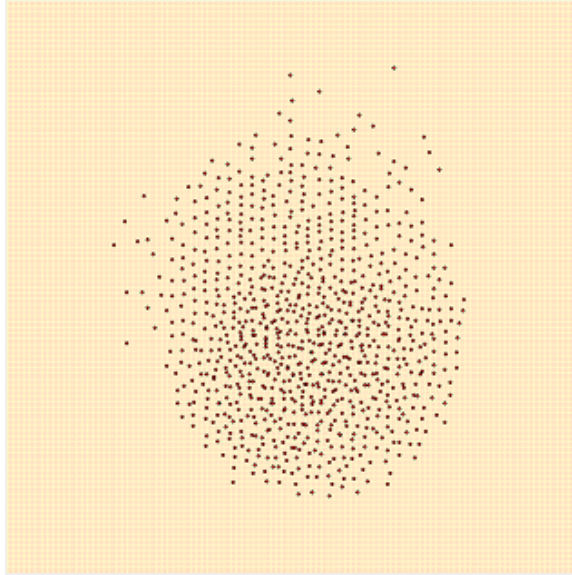
Figure 4. Visualization of simulation in progress

Average time to detection is an important goal, and an objective mechanism by which to study an algorithm's effectiveness. This is calculable from tiles visited per hop, e.g., if coverage after initial deployment is 56%, then the average time to detection is 0 for 56% of the tiles, and for 44% is determined by time for a sensor to get to the tile.

The following data shows the coverage resulting from numerous simulations using the WGB algorithm, varying the size of the coverage field and number of hops per sensor, for a fixed number of sensors. We performed considerable experiments and made comparisons between the WGB algorithm compared to sensors that randomly pick an adjacent tile until they exceed mobility constraints, as well as an algorithm where sensors randomly pick an adjacent unoccupied tile until mobility constraints are exceeded, and WGB performs better (visits more tiles) than either of the random algorithms for Gaussian deployment around a point.

7.1. Coverage for various sized coverage areas . Because sensor deployments can result in concentrations of sensors, and with mobility being limited,

it could require more sensors or more hops per sensor in order to assure complete coverage. The data presented in Figure 5 shows the average level of coverage resulting from numerous simulations using the WGB algorithm, varying the size of the coverage field and number of hops per sensor, for a fixed number of 100 sensors. Note: Hops0 indicates the initial coverage at deployment time using Gaussian distribution around a point.
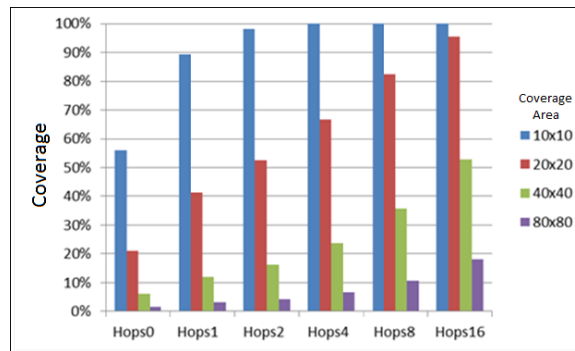
Figure 5. Average # Tiles Visited - Varying Coverage Area

As shown, as we increase the hops per sensor, the resulting coverage increases. With a 20x20 grid (400 tiles) and 16 hops, these 100 sensors cover the field, but due to the Gaussian concentration of sensors around a point, some sensors had to travel a considerable distance to visit outlying tiles.

7.2. Coverage for various number of sensors . When we vary the number of sensors, we can also see a consistent improvement in the coverage. For the next set of experiments, we fixed the size of the coverage field to a 30x30 grid, and again used Gaussian deployment around a point to produce the results in Figure 6. However, this time we vary the number of agents deployed and examine the resulting coverage performance, both with agents that are allowed to hop once, and agents allowed to hop twice.
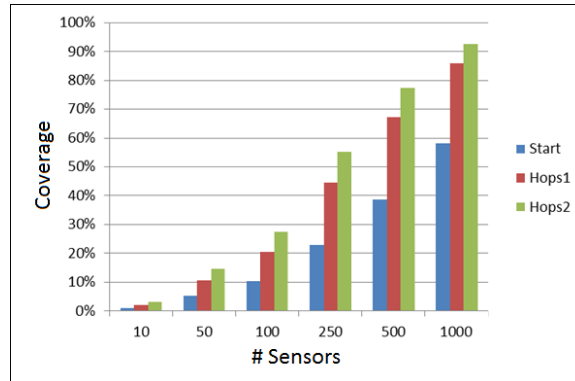
Figure 6. Average # Tiles Visited - Varying Number of Sensors

As shown in Figure 6, it took nearly 1000 sensors hopping twice each to achieve an average of 90% coverage, thus attempting to overcome a fixed deployment concentration using increased number of sensors could be less practical than varying other parameters.

7.3. Coverage when the deployment concentration (σ) varies . Our deployment implementation uses Gaussian sets of coordinate values that produces numbers with a mean of 0, and a standard deviation of 1. That is normalized, scaled using value $w * \sigma/2$, where w is the width/height of tiles in the grid. The lower the value for σ, the less concentrated the initial deployment. For this experiment, we looked to see what happened when we vary the value of σ and analyze the resulting coverage performance. Figure 7 shows coverage achieved for various values of σ and number of hops. As shown, the less concentrated the initial deployment, the greater the value from additional mobility, although at these less dense concentrations, there is a diminishing return for additional mobility, while the tighter concentrations continue to crave additional mobility to improve coverage. In other words, additional mobility is required when the deployment is more tightly concentrated to achieve the same measure of success. Figure 8 shows coverage achieved with various density and concentration values, fixing the number of hops at 4. With a value of $\sigma = 0.5$, we

improve an initial value of approximately 33% coverage an increase in nearly 60% to a coverage achieved with 4 hops of over 90% coverage.
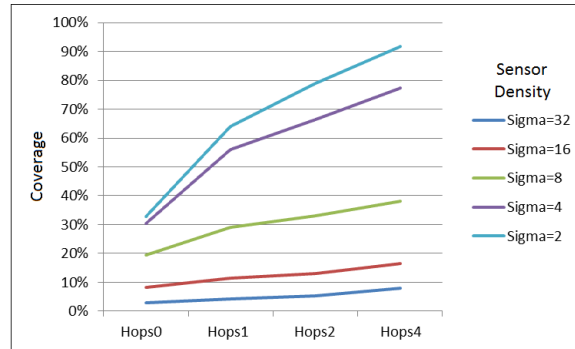
Figure 7. Average # Tiles Visited - Varying Deployment Concentration
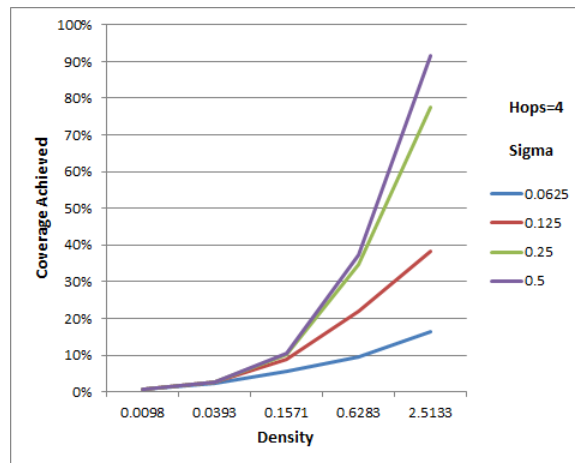
Figure 8. Average Coverage Varying Concentration and Density

7.4. Effects of mobility on coverage performance . Using the data from the experiments above, we would now like to examine the effects of mobility on coverage. We will do this using a metric $e = H/\%$ where H is the number of hops taken by sensors, and e is the hops incurred per percentage increase in coverage. In other words, how much mobility is expended to achieve additional levels of coverage in a variety of scenarios. Figure 9 shows the data for this metric when varying the

size of the coverage area. Continuing the point made in the previous section, for less concentrated deployments, there is a diminishing return on additional mobility as compared with a steadily growing improvement in coverage at tighter deployment concentrations.

What we conclude is that the effects of allowing a measure of mobility on the achieved coverage success are greater in more sparse deployments than in concentrated deployments. The results illustrated in Figure 9 are intuitive considering that as sensors become more concentrated, the initial coverage overlap increases, sensors must travel further to find as yet untrodden ground, additional time is (therefore) required to achieve the same coverage as less concentrated deployment scenarios, and additional mobility is required for sensors to overcome the initial deployment.
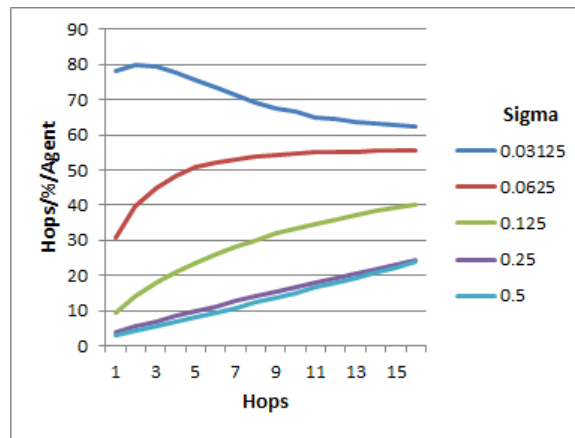
Figure 9. Mobility Per % Coverage Increase - Varying σ

On the other hand, increasing mobility adds to the quality of coverage, but with a diminishing return on investment. Results were consistent for scenarios where the number of sensors and the values for σ were varied, which was likewise expected.

8. CONCLUSION

Practical algorithms for employing networks of unmanned, autonomous mobile devices with limited mobility, in sparse regions where it is crucial that events are

located, identified, and communicated is approaching ubiquity. As sensor technology improves and becomes more mobile, computing power expands while requiring less and less energy, and production of drones and robotic platforms become more robust, algorithms for directing the behavior of these devices will only become more of a focus.

We have explored a variety of problems related to coverage of a coverage field. Within this we have focused on the problem of Exploratory Coverage with sensors that have limited mobility capabilities. We believe the taxonomy of the various subproblems within Exploratory Coverage should prove helpful for further study to allow focus on certain facets of the problem.

A range of prior work exists and has provided a wealth of conclusions from which to inform our work in the area of the different approaches to the problem of Exploratory Coverage, and we have shown several algorithms for finding approximate solutions.

We then design a purely localized and distributed approximation algorithm and provide simulation results. These simulations are conducted to demonstrate the effects of limited mobility on area coverage. Here we show that the effects of mobility constraints are more pronounced in sparse deployments than when sensors are deployed in a more concentrated fashion.

PAPER III. FAULT TOLERANCE IN AREA COVERAGE ALGORITHMS FOR LIMITED MOBILITY SENSOR NETWORKS

1. ABSTRACT

In sparse deployments of mobile sensors, the mobility of sensors is required to search the coverage area in an attempt to achieve polling—complete, possibly repeated, area coverage over time. Mobile sensor platforms are vulnerable to a variety of hazards during normal operation. When sensors lose the ability to mobilize due to mechanical failure, environmental factors, or simply exhausting their energy source, coverage effectiveness can be seriously impacted. Ideally, algorithms should adjust their behavior to compensate for failure modes in order to avoid areas statically covered by disabled sensors as well as adjusting their behavior to cover the areas assigned to sensors that are no longer able to mobilize. In this work we demonstrate the effects of disabled mobility on area coverage algorithms due to their inability to adjust behavior, and suggest mitigation strategies and the impact on improving coverage in the face of disabled mobility.

2. INTRODUCTION

There are numerous factors driving increased attention to usage of automated sensor drones that consist of a hardware platform with onboard command/control, sensors, and effectors that provide for mobility. Human safety factors have long been a primary motivator for interest in employing robots for a variety of tasks where humans would prefer not to be. Also, mobile sensor hardware platforms have seen many recently advances such as higher computation power, relatively lower weight, and lower power requirements that allow drones to carry much more computational

capacity and payload or stay deployed and functional for longer periods of time. More and more, these platforms are more readily available, as the benefits of mass production of commercially designed systems are realized.

Mobile sensors are physical devices that are subject to observable failure rates. One of the most common problems for mobile sensors is that the effectors that provide for the mobility function of the platform fail [43], or that terrain or other issues cause the mobile sensor to become stuck while all other features of the platform continue to function as normal.

It is intuitive that one requirement of an algorithm being analyzed is that for the algorithm to function, sensors must be aware to some degree of the relative position and mobility restrictions (failure modes) impacting other sensors, otherwise there is no way they can (nor any reason for them to) alter their behavior. Mobile sensors need not support full localization, however. Awareness of the failure modes of other sensors allows the network to be fault-tolerant, self-healing, and to dynamically change from initially homogeneous to heterogeneous with respect to mobility as agents adopt different roles (whether by choice or as observed).

Further, we acknowledge that various subsystems of a wireless sensor exhibit different energy requirements than others. Since our primary priority is to maintain a given network quality of service, we are required to utilize the mobility feature of the sensors. What we will establish is a schedule, whereby a certain number of sensors lose their mobility but retain their other functions, and will analyze the extent to which other sensors are able to compensate for this failure mode by altering their movement strategy to preserve the required quality of service while avoiding the area covered by the disabled sensors.

Given an objective measurement for lifetime and effectiveness of a sensor network, we explore the effects of disabled mobility on these metrics. From this work, we can see that as sensors lose their mobility, they become, in essence, static rather

than mobile sensors. When the deployment and/or the algorithm that governs their behavior ensures that mobile sensors are spread sufficiently to adequately cover the search area, then coverage impact can be minimal. Conversely, when the deployment is concentrated or the algorithm fails to spread sensors prior to many of them failing, coverage effectiveness is vulnerable. Additionally, when mobility fails but other functions such as communication continue to operate, a sensor can communicate misleading intentions to other sensors. In essence, this can have outcomes similar to an attack where blind spots are created. The disabled sensor cannot navigate to the intended location to cover it, and other mobile sensors choose not to go there because they believe it is already covered.

In this paper, we examine related work in the areas of limited mobility, reliability analysis, and fault tolerance. After a brief survey of mobile sensor platforms, their mobility limitations, and their vulnerabilities to device failure, we describe the problem of assessing the impact of disabled mobility on coverage algorithms. We define the reliability model we will use, describe the simulation platform and parameters selected for our analysis, several coverage algorithms that are used for comparison, and then present findings, conclusions, and future work.

3. RELATED WORK

Fault tolerance in mobile sensor networks, the topic of algorithms that behave in a way that tolerates failure modes while still cooperatively pursuing a goal, can be found [44, 45, 46], but the specific topic of fault tolerance with respect to limited mobility sensor networks and how failures affect coverage performance has not.

One area where limited mobility affects performance of a mobile sensor network is when sensors are deployed for blanket coverage. The lifetime of such networks, and algorithms for preserving/extending the lifetime, has been extensively studied.

An example uses redundant sensors in a dense deployment so that as sensors fail other sensors in the same region can wake up and take over for the missing sensor [47]. There are formulae that can be used to objectively assess the expected lifetime of a network of sensors [48]. Also, many works have explored the idea of optimized message routing for sensor networks in the event that some sensors fail, so that messages can be routed through other paths [49]. However, there has been little work devoted to the area of exploring what happens when a sensor continues to function even though it is no longer able to navigate. This is a real concern given that in many real world sensor platforms, energy requirements for mobility account for a large portion relative to that required for sensing and communication. The closest analogy is the study of hybrid sensor networks, where the term hybrid refers to the fact that sensors are non-homogeneous: some are static, and others are mobile [16, 50]. However, no works were found that examine the problem of sensors failing according to a predetermined failure model, additionally that the failure is limited to the mobility feature of the device, and relates this to the impact on coverage effectiveness.

As further background, a number of works examine sensor network lifetime from a rather fatalistic point of view, expressing desire to describe and understand an inevitable upper bound on the utility of a network of sensors [51].

It was shown that effectors–devices that perform actuation (including mobility), such as the motor, appendages, treads/wheels, and related connections—was observed to account for 35% of mobile robot failure, the largest single reason for failures [43]. This makes consideration of the problem of maintaining coverage quality of service (among other goals) using cooperating sensors in the network, an important aspect of mobile sensor network research.

Various works explore fixed deployments (no mobility following the initial deployment of the sensors), where mobility is not a concern toward energy constraints on the lifetime of the network. In a dense sensor deployment scenario, more sensors

are deployed than are required to cover an area, and when a sensor fails, other sensors use a protocol to decide which sensor wakes up and takes the place of sensing the missing area. In some cases, sensors have a limited ability to exercise mobility, and can move closer to the hole in coverage in order to adjust for the missing sensor.

4. RELIABILITY IN SENSOR PLATFORMS

Numerous mobile sensor platforms have been in development in recent decades, including ground-based, lift-based, buoyancy-based, and space-based. Ground-based platforms (sometimes referred to as UGV's [52]) can be tiny weighing only a few centimeters/grams, using battery-powered micro circuitry, or as large as automobiles weighing tons using internal-combustion engines. These platforms are subject to a variety of mechanical failures, are vulnerable to obstacles found on the ground in the environments in which they operate, as well as terrain variations and pitfalls. The energy source (weight and conservation) is a major factor limiting the mobility of these platforms.

Lift-based, or aerial, platforms are devices that employ the physics of lift in order to remain in a state where controlled mobility is possible. The size of these devices can range from small, hand-held devices, up to large military/commercial aircraft. Identifying aircraft with a low Reynolds number provides a way to construct devices that are useful for lab research [1]. The Reynolds number can be expressed as shown in Equation 3, where ρ is the air density, L is airfoil length, υ is velocity, and μ is the viscosity of the substance through which the device moves.

$$Re = \frac{\rho L \upsilon}{\mu} = \frac{\rho \upsilon^2}{\frac{\mu \upsilon}{L}} = \frac{inertia}{viscosity} \tag{3}$$

Utilizing Equation 3 allows researchers to create small, lightweight devices that can move slowly and stay aloft for longer periods of time. A challenge faced by

lift-based platforms, however, is that they must expend energy to maintain continual lift. Mechanical failures are often fatal due to engineering the devices to use minimal structural material to minimize fuel requirements and allow for more payload, which in turn makes the devices more fragile than their ground-based cousins.

Buoyancy-based platforms solve many of the problems faced by ground-based and lift-based platforms. Examples of this type of platform include blimps and boats. These devices can be tiny to enormous commercial tanker ships. They are characterized by the ability to maintain a stable navigational state for long periods of time (indefinitely, barring other issues such as leaks), and the ability to support a much larger payload over time than lift-based or even ground-based platforms. Vulnerabilities include currents in the substance (typically water or air) in which the devices operate, weather, and obstacles. When we consider the relative densities of substances, we can approximate the weight and buoyancy for devices utilizing these substances as shown in Table 1. Thus the desired payload can be defined and the device characteristics tailored to fit.

Space-based platforms have been in use for nearly six decades. These devices must use some combination of lift, buoyancy, and thrust in order to place the device in "space" where it is capable of remaining aloft in a state where its navigational attributes are governed by inertia and orbital mechanics, and where the viscosity becomes negligible. Such platforms are vulnerable to impacts with other objects traveling at very high velocities, orbital decay causing atmospheric reentry, cosmic rays and radiation, extreme heat and cold, in addition to standard mechanical failures with rare opportunities for service/repair.

Reliability analysis studies the probability of devices performing the function for which they were designed over a period of time within specified parameters. In [53] we see analysis of failure rate models for devices. We describe time-to-failure as a probability density function (PDF) or cumulative density function (CDF). This may

Table 1. Approximate weight and buoyancy of various substances

Substance	Weight	Buoyancy
Air	$1.2256 Kg/m^3$	—
Hydrogen (H)	$0.0857 Kg/m^3$	$1.1399 Kg/m^3$ (H v. Air)
Helium (He)	$0.1691 Kg/m^3$	$1.0565 Kg/m^3$ (He v. Air)
Water (H_2O)	$988.2 Kg/m^3$ @ $20\,°C$	$0.24875 Kg/m^3$ (Air v. Water)

be expressed mathematically as shown in Equation 4. The function $f(x)$ represents a distribution of failures over time, and the interval between t_0 and t_1 is the period of time during which the devices are observed.

$$R(t) = Pr\{T : t_0 \leq x < t_1\} = \int_{t_0}^{t_1} f(x)dx \qquad (4)$$

Failure rates for these various platforms are becoming more widely available as technologists spend more time settling on one design and tracking its reliability [43, 46, 52, 54, 55, 56]. As these platforms become more common, we will be able to develop more applicable and accurate failure rate models for each type of platform.

5. PROBLEM DETAILS

We define the coverage field as a region that is observed as a plane to sensors. A set of mobile sensors is deployed using a deployment function. In this paper, we focus on two deployment schemes. First, a purely stochastic means of evenly distributing sensors throughout the field. Second, a stochastic method that produces a Gaussian approximation of a Poisson distribution around a point, as if the sensors might have been dropped from an aircraft and dispersed organically at various distances and orientations relative to the drop point.

We focus on sparse deployments in which the number of sensors n is defined in Equation 5, where n is the number of deployed sensors, A is the area of the coverage

field and r is the sensing range. This relationship ensures that the number of sensors being lower than required to make blanket coverage possible. This allows us to focus on finding solutions to the problem of polling–minimizing detection time for any events in the coverage field, while maximizing the number of times we can poll all points in the coverage field over a given period of time.

$$n < \frac{2A}{3\sqrt{3}r^2} \tag{5}$$

In balanced deployments, the problems shift from finding solutions that minimize the time to achieve (and maintain) blanket coverage, whereas in dense deployments the problems shift from the challenge of providing polling to one of maintaining blanket coverage or redundant blanket coverage. When a sensor becomes mobility-disabled in balanced to dense deployments, the network's ability to maintain blanket coverage for a length of time can be shortened as sensors ultimately fail completely and the inability of other sensors to take their place causes coverage holes.

The mobility of the sensors is considered to be limited, in that there is a probability that at a certain time interval from the drop time that a given sensor might suddenly lose the ability to move. This simulates the lifetime of the mobility feature of the sensor. The sensor continues to be able to take measurements of its environment from this location, however. The number of sensors that have failed over time is controlled such that it follows a probability density function. As an example, the number of sensors that have failed over time might look like one of the models shown in Figure 1.

The "bathtub curve" model for failure rates has been described [53]. In this model, numerous initial failures are observed, followed by a stable period where few failures occur, and finally a period of time where devices succumb to the useful lifetime of any of a number of their components causes a relatively higher failure
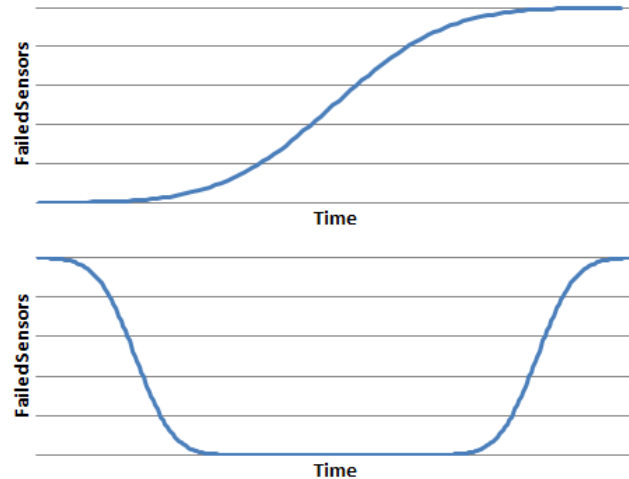
Figure 1. Cumulative distribution function (top) illustrating how we will distribute the disabling of mobility of sensors over time, plus Bathtub Curve (bottom) showing another well known failure rate model

rate to account for a majority of the remaining devices. While this model describes the failure rate of an entire population of devices over time, this model may not be as useful for studying the effects of failures of a specific set of devices in use in the field. This is due to the fact that initial testing and quality control measures will identify defective products prior to deployment, and devices may be replaced in the field long before they actually fail during use. For this reason, we focus on the cumulative distribution function (CDF) model for our simulations that assumes no failures initially, but a growing number of failures as the simulation progresses, followed by a few sensors that fail later.

6. ALGORITHMS

In our simulations, we chose several algorithms to analyze and compare with varying sensor count (sparsity), two deployment schemes (random and Gaussian around a point), with and without applying the schedule of disabled mobility, and in

some cases with and without modifications to the algorithm to mitigate the effects of disabled mobility.

The Random Walk and Random Direction Walk [57] are used. The Random Walk algorithm has sensors simply choose at each opportunity in time a random direction to move, whereas the Random Direction Walk algorithm chooses a random direction up front and continues that direction for the life of the simulation. Although these algorithms include no cooperative features, nor do they attempt to avoid gaps or redundant coverage in any way, they provide a good baseline for comparison to other algorithms.

The Proxy [58] and WGB [59] algorithms were also used. The distributed Proxy-based algorithm involves both static and mobile sensors which bid for new locations in order to heal holes in coverage. The WGB algorithm, also a distributed heuristic-based approach, uses an internal tile-coloring model to merge data about what areas nearby are not covered (white), have sensors nearby that could cover (gray), and areas that are already occupied (black), in order to identify the location of highest need. We focus on WGB in order to eliminate the variable of static sensors from disabled mobility sensors. Other algorithms such as Virtual Force (VF) [30, 60] were also examined.

7. EFFECTS OF DISABLED MOBILITY

We anticipate a few challenges that will affect coverage efficiency in the face of disabled mobility. First is the fact that a disabled sensor is sitting in one place sensing the area around it and other sensors that pass through this area will duplicate coverage resulting in a loss of efficiency. Second, the disabled sensor could have been expected to have covered a portion of the field itself had it not become disabled, thus other sensors may need to adjust their movement plan in order to cover the

area excluded by the loss of the disabled sensor. Another factor of interest is the probability of detection and the detection time delay behavior in the presence of the disabled mobility schedule.

With some algorithms, disabled sensors may in fact mislead other sensors about their intended mobility plan and affect coverage in ways that would not be seen if the sensor were to completely fail.

Let us consider a scenario where we assume a random distribution of sensors across the search area, where sensors have unlimited mobility (range). The sensing distance is configured so that this is a sparse to balanced deployment (i.e., the ratio of sensor range to number of sensors relative to search area precludes blanket coverage). The goal is to maximize area coverage over time (or synonymously to minimize detection delay). We examine two reference algorithms. First, the Random Walk algorithm, where each mobile sensor starts exploring the area in random moves. Second, we examine the Random Direction Walk algorithm, where each sensor begins by picking an initial direction and continually moves straight in that direction indefinitely.

Analyzing the results of a schedule of sensor mobility disability surfaced a challenge with this scenario. Examining the initial deployment, we observe a random distribution of mobile sensors throughout the search area. Also, at any time in the future, a snapshot of the region also shows a distribution with no less random features than the initial deployment. Despite the fact that the point at which a given sensor becomes disabled is according to a pre-determined schedule, the location at which it resides when it becomes disabled is again no less randomly distributed than the initial deployment. Thus observing simulation of this scenario over time as seen in Figure 2 shows that although the performance isn't great at any point in time throughout the runs, disabling the mobility of the sensors doesn't hurt the algorithm in an interesting or unexpected way.
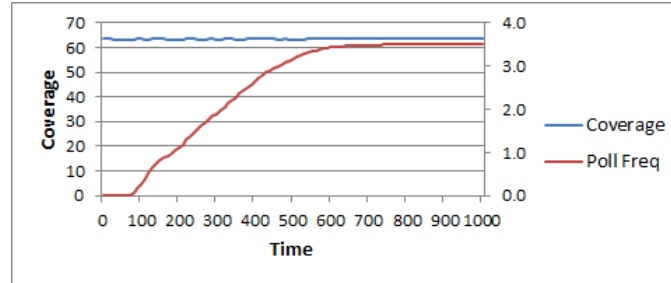
Figure 2. Random walk coverage and polling frequency over time

Using the Random Direction Walk algorithm produces analogous results, and both algorithms are consistent even when the number of sensors is varied. Figure 3 shows a consistent drop-off in polling frequency with random direction walk across a variety of sensor counts. Polling frequency eventually flattens due to the sparse deployment density and the fact that disabled sensors fail to iteratively cover the area over time. This produces an equivalent increase in average detection delay as more and more of the area must be polled by a decreasing population of sensors with mobility.

Figure 3. Random direction walk polling frequency over time

When we examine the effects of disabled mobility on the WGB algorithm, we see a consistent drop in coverage performance, and falling to as much as 20% loss of

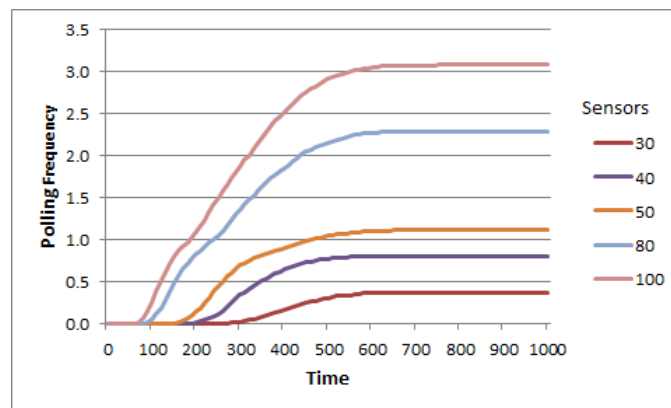coverage as sensors begin to lose mobility. Figure 4 illustrates how many percentage points coverage drops with the WGB algorithm in particular, simply by adding the schedule of disabled mobility over time. This is actually quite good considering that the algorithm pushes sensors from their initial deployment to a balanced coverage of the area rather quickly.
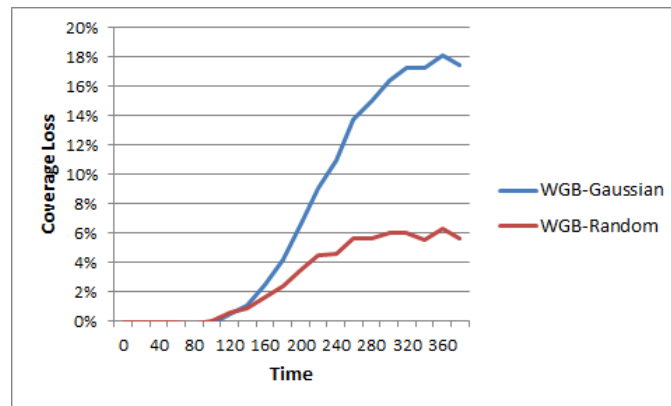
Figure 4. Percentage points lost adding disabled mobility to WGB algorithm.

Given the size of the coverage field in these simulations relative to sensor range and number of sensors, we can see the computed maximum coverage achievable for various sensor counts in Table 2. These are upper bounds, and rely on none of the sensors overlapping areas covered by other sensors. In practice, for our distributed algorithms to consistently achieve polling, the number of sensors remains at 80 or below.

We can also see from Figure 5 a representation of the performance of the WGB algorithm in terms of the coverage percentage over time for a varying number of sensors. In order to properly interpret the sparsity of the deployment given the specified sensor counts, we refer once again to Equation 5 with a configuration of coverage area and sensor range that results in a balanced deployment would require a value of $n \approx 76.98$.
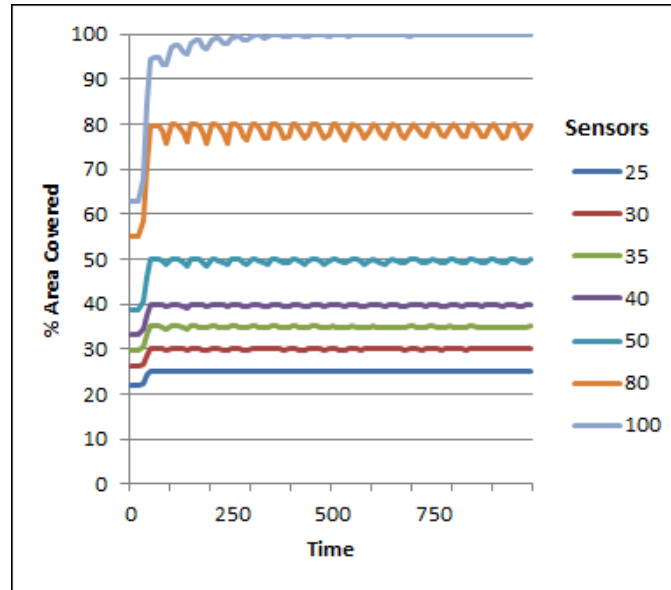
Figure 5. WGB algorithm, coverage area over time varying sensor count.

Regarding the problem of disabled mobility causing potentially misleading information being broadcast to other sensors, we see that there is a clear impact to coverage efficiency. Figure 6 shows that as more sensors become disabled, there is a degradation to the coverage percentage over time that can reduce coverage by as much as 15%.

By making a small adjustment to the WGB algorithm to detect this failure mode, plus a behavior change when the failure mode occurs such that sensors refrain from broadcasting an intention to move that will not occur, we see an improvement in both Gaussian and Random deployment modes of a full percentage point. Figure 7 shows the improvement for one configuration. As shown, the improvement begins as sensors start to fail. When a growing number of sensors are unable to move, the ability for the WGB algorithm to continue to cooperatively explore the coverage field without gaps or significant redundant coverage becomes apparent as compared to Random Walk, Random Direction Walk, and other algorithms.

Table 2. Max coverage % for various sensor counts.

Sensors	Max %
25	39.26
30	47.12
35	54.97
40	62.83
50	78.53
80	125.66
100	157.07

8. CONCLUSIONS

One thing we can observe from these results is that coverage algorithms that do a good job of quickly reaching a desirable location from their initial deployment, cooperate to avoid gaps and redundant coverage, and continue to leverage what mobility is available throughout the sensor network, produce better results as sensors lose their mobility than algorithms that rely on statically touring or other more methodical means of exploring and covering their environment. The WGB algorithm, for example, saw only minimal degradation of coverage quality of service, and performed well in sparse to balanced deployments in the face of a schedule for disabled mobility.

With extremely sparse deployments, sensors that are mobile come into contact with disabled sensors less often. In these scenarios, we observe through simulations that algorithms such as random direction walk have less of an impact than algorithms that tour an established territory, because in the latter case, once a sensor becomes disabled, there is no sensor to cover that sensor's territory. As the deployment becomes less sparse, algorithms that try to avoid one another are more vulnerable to
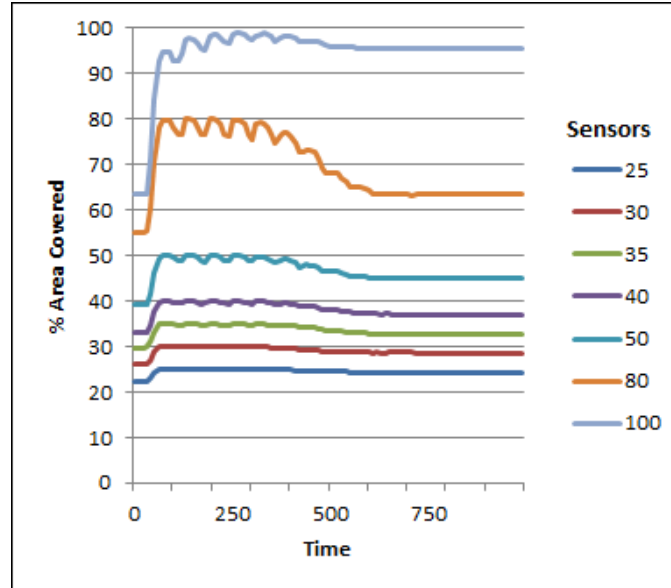
Figure 6. WGB algorithm with disabled mobility, coverage area over time varying sensor count.

being misled by disabled sensors that continue to broadcast their intentions to move, but never do.

The disabled mobility problem has particular significance because, because as defined, the outcome can be demonstrated clearly as an extension of prior proven simulation techniques. The proposed approach introduces a factor whereby sensors become immobile at various rates over time. When the coverage algorithm is to pick a random direction, then sensors will disregard the location and coverage provided by disabled sensors and will proceed to duplicate coverage. When algorithms avoid those areas, coverage effectiveness can be shown to increase. As more sensors become disabled, coverage becomes degraded, as we have shown.

Figure 7. Percentage improvement in coverage efficiency by detecting failure mode.

9. FUTURE WORK

Communication protocols have been extensively studied from a number of perspectives. However, there is potential for augmenting these protocols to transmit failure modes along with existing packets in order to allow distributed algorithms to reactively modify their behavior to make the sensor network self-healing fault tolerant.
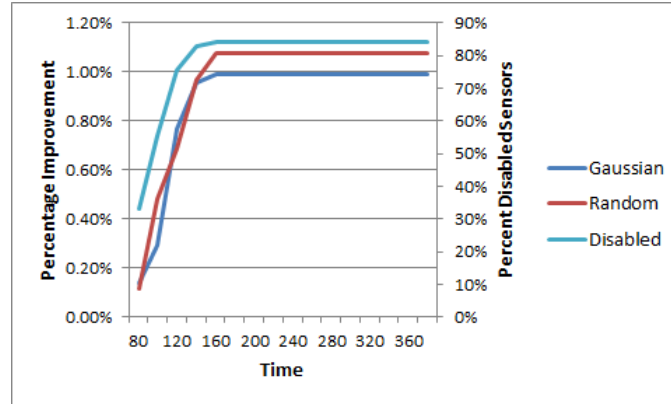
SECTION

2. SENSOR ANALYSIS SIMULATOR

For many of the simulated results referenced within this work, the *Sensor Analysis* simulation tool was used. This program was developed by the author and is described below. Source code for this project is available as a public repository on the website GitHub (http://github.com), a code-hosting repository based on the Git version control system [61]. Detailed information and updated software versions can be found on a page devoted to the Sensor Analysis project (http://marksn-ms.github.io/SensorAnalysis). This means of publishing the source code was done in order to allow future work to continue on the project with hope that advances can result due to the phenomenon of knowledge crowdsourcing [62].

2.1. ARCHITECTURE

The tool is written in the *C#* programming language using the Microsoft Visual Studio® development environment and is designed to run on the Windows™ operating system. The program WorldSim.exe presents a visual representation of the search area, which can be represented as a grid of rectangle or hexagonal tiles. Specification of simulation parameters, as well as starting, stopping, and monitoring, are performed through the client.

The core design of the application relies on classes and interfaces defined in WorldSim.Interface.dll. Base classes/interfaces include Deployer, Incident, Inhabitant, Marker, Message, SelectableObject, Tile, and Watcher. These objects and their purpose are shown in Table 2.1.

The WorldSim.Interface.dll module (or "assembly" in .Net parlance) also contains several common helper objects. One such object is an adaptation of the ECRandom[63] class that provides a member function that returns values of data

type Double in a Gaussian distribution between -1.0 and 1.0. This class provides the mechanism used by the GaussianDeployer class (an implementation of the Deployer interface) and the RouletteWheel class.

Table 2.1. Common interfaces found in WorldSim.Interface.dll and their purpose.

Class/Interface	Purpose and description
Deployer	Implementations of this interface are used to deploy objects in the simulation to desired locations.
Incident	An object being sought by agents.
Inhabitant	Represents a simulated agent.
Marker	Useful as a reference point, such as may be used by Ant Colony Optimization (ACO) agent implementations.
Message	An object that can be delivered to other agents, towers, or base stations using the messaging subsystem.
SelectableObject	A key interface, this represents required functionality for any class that can be displayed, selected, deployed, etc.
Tile	Represents a component of the search area.
Watcher	Implementations of this interface sit by and monitor from step to step what is happening in the simulation and often record and log results.

The RouletteWheel class provides a way to perform Monte Carlo selection[64], a helpful technique in many scenarios where the likelihood that an item in a list is chosen is relative to some weight factor. One simulated agent was developed, for example, using Monte Carlo selection to slightly favor the current navigational

heading over a new heading so that the agent tends to continue moving in the same direction.

2.2. USER INTERFACE

Starting a simulation from the graphical user interface (GUI) is straightforward. The analyst first creates an environment using "File | New" in the menu (or Ctrl-N on the keyboard), specifying the number of rows and columns of tiles, the height and width of each tile, the shape of tile (currently rectangle and hexagon are implemented), and clicking OK. An example of a simulation that has been created is shown in Figure 2.1.

At any time, an object's properties can be examined (and in some cases modified) through the user interface. Figure 2.2 shows a simulation that was created using a grid of hexagon tiles where a number of objects have been deployed. As shown, one of the objects in the simulation has been selected and its properties are displayed in the pane on the right. Those properties shown in bold face text can be modified. Tiles and objects derived from Inhabitant and Incident can be selected using the mouse by clicking on the object in the simulated environment.

Next, sensors or other objects are added to the environment. This is done using "Edit | Add objects" in the menu (or Ctrl-A on the keyboard), selecting the agent type from the menu of available choices, the number of objects to add, and clicking OK. Custom properties, if any, supported by an agent type can also be specified here, and all objects added to the environment will reflect the desired property values.

By default, all objects are added at the same location, the center of the first tile. In order to support a variety of deployment methods, the program was designed to allow custom Deployer objects to be utilized. In order to invoke a Deployer, the analyst uses the "Edit | Deploy" menu item (or Ctrl-D on the keyboard), selects the
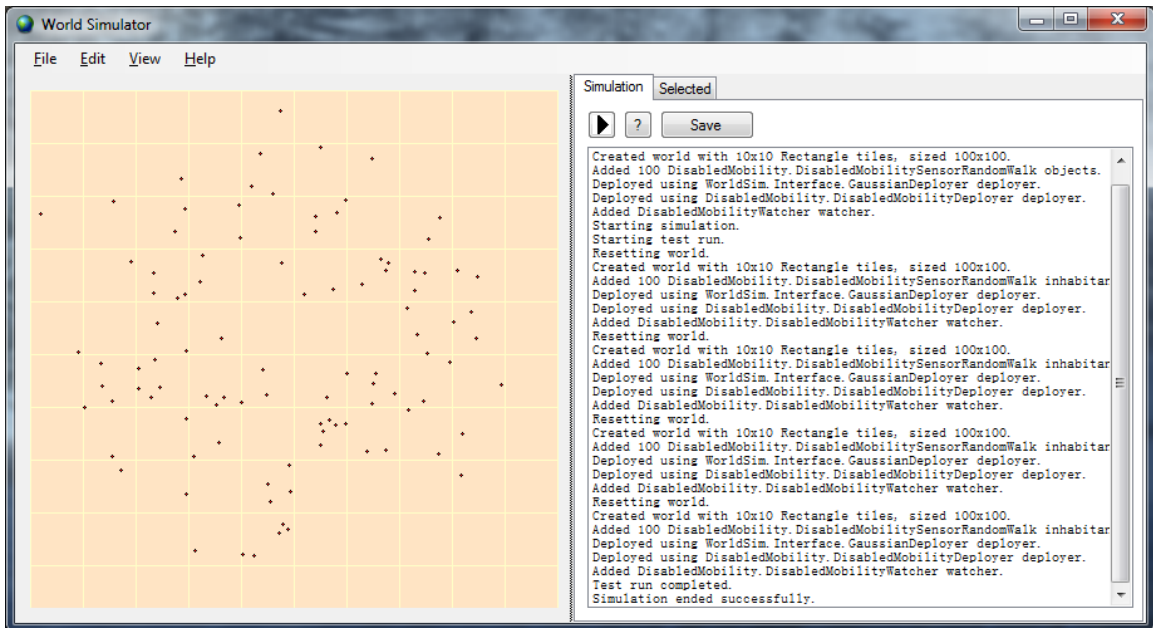
Figure 2.1. Basic deployment of sensors on square grid search field and showing simulation setup sequence.
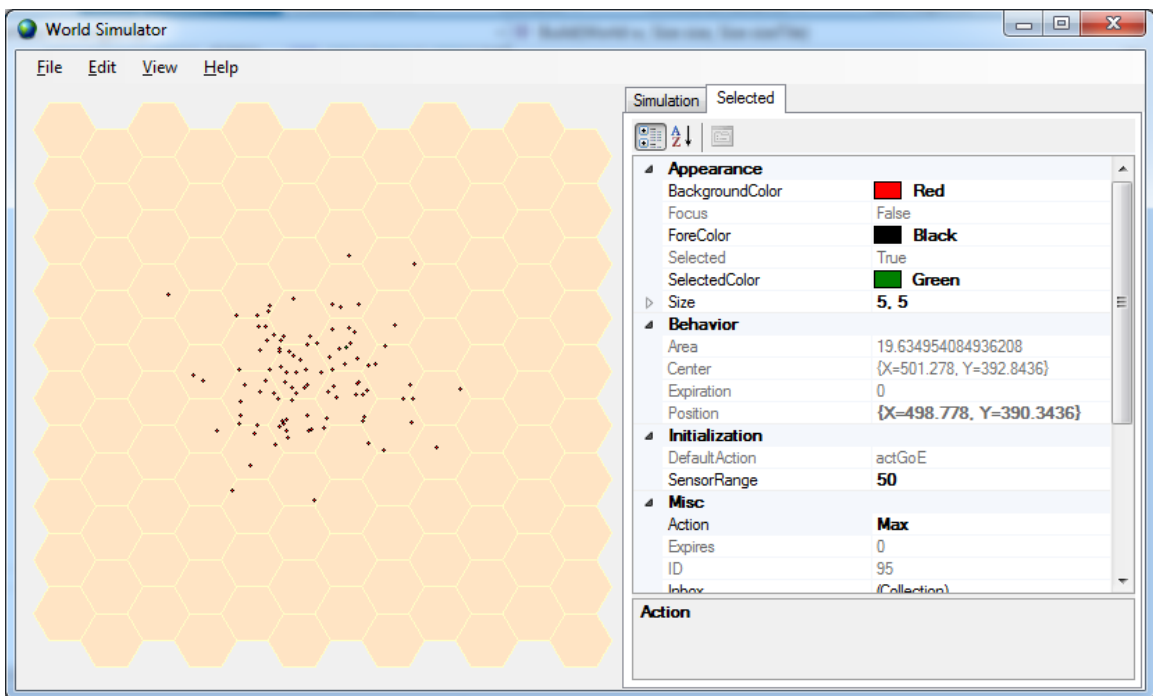

Figure 2.2. Basic deployment of sensors on hex grid search field and showing selected object properties.

type of deployer, and clicks OK. Standard derivations of the Deployer class that are currently available include Random and Gaussian. The Random deployer will find any object in the simulation whose class is derived from Inhabitant and move it to a pseudo-random point that is evenly distributed across the simulated environment. The Gaussian deployer (short for Gaussian-around-a-point) finds the center of the simulated environment and moves all objects whose class is derived from Inhabitant to points distributed in Gaussian fashion around that point.

Deployer objects can also be used to apply dynamic properties to certain objects in the environment. For example, the Disabled deployer finds all objects in the environment whose class is derived from DisabledMobilitySensor and sets a time at which mobility for each object becomes disabled. This is used extensively in Paper 3.

In order to monitor a simulation using methods common and useful to many exercises but also to allow for extensibility, the Watcher class was created. Watcher objects can be added to the simulation using the menu item "Edit | Add Watcher" (or Ctrl-W on the keyboard). The World class fires events at significant points in the simulation life cycle, such as before a simulation starts, after it ends, and prior to ticks of the world clock. A Watcher derived class implements behaviors for events of interest and is able to access information about what is happening in the simulation. One example of this is the DisabledMobilityWatcher that writes a string to a log file every 20 time units (ticks of the clock) that contains the coverage progress for analysis following the conclusion of the simulation.

Figure 2.3 shows the life cycle of a test run. This can be initiated from the user interface by selecting "Edit | Go" (or F5 on the keyboard), specifying the number of experiments, the time units for each experiment, and clicking OK. The program will run the test until completion and indicate headway using a progress bar as well as status messages. At the point in the life cycle where the world is reset, a number
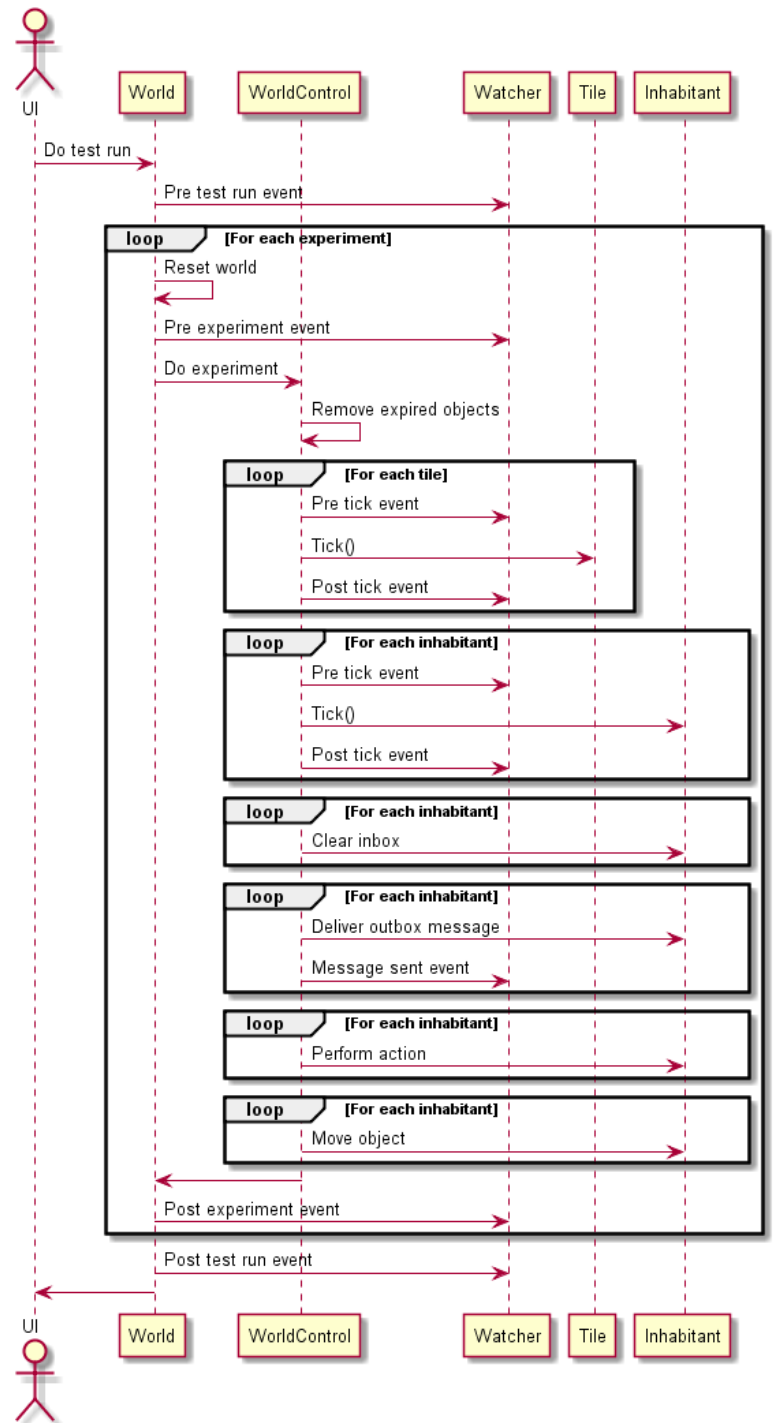
Figure 2.3. Sequence diagram of test run life cycle.

of operations occur. One such action is to recreate the search environment in the dimensions specified either in the GUI or options file. Another action is to add Inhabitant, Incident, Deployer, and Watcher objects in the quantity and order that they were specified. In this way, each test run conducts experiments using the same options.

Many of the simulations were run on a personal computer (PC) with an ASUS P7P55D Pro motherboard, Intel Core i7-870 Processor running at 2.93GHz, 16GB RAM, and SSD hard disk.

The program was written to support execution using command-line parameters, making it ideal to run many tests in parallel and repeatably using a batch file. An example of a typical command execution would look like this:

```
worldsim.exe /options "Parameters.txt" /go /exit
```

This specifies a parameter file is to be read that contains the simulation options, and also directs the program to begin the simulation upon startup and close following completion of all experiments.

A typical file would contain options in XML format such as those shown in the sample file in Figure 2.4.

```xml
<?xml version="1.0" encoding="utf-8"?>
<TestSettings>
  <Title>Sample Test Run</Title>
  <Environment TileShape="Rectangle"
               TilesWide="10" TilesHigh="10"
               TileWidth="100" TileHeight="100" />
  <Instructions>
    <string>Inhabitant, count:80, type:DisabledWGB</string>
    <string>Deployer, count:1, type:Gaussian</string>
    <string>Deployer, count:1, type:Disabled</string>
    <string>Watcher, count:1, type:DisabledMobilityWatcher</string>
  </Instructions>
  <Duration>2000</Duration>
  <Repeats>100</Repeats>
</TestSettings>
```

Figure 2.4. Sample options file specifying parameters for a test run.

As a benchmark, a test run using the parameters in Figure 2.4 runs on the hardware listed above in about an hour. Many such test runs can be run simultaneously as allowed by available memory and CPU capacity.

2.3. EXTENSIBILITY

The tool was designed to be extensible by using a common core application and rendering engine while allowing for standalone assembly modules to be implemented and used for widely varying test scenarios with minimal code in a small amount of time. In order to create an assembly, a researcher need only implement a small number of interfaces (as described below) and add the assembly information to the application's configuration file which is loaded at runtime.

Provided are sample implementations of a sensor, a deployer, and a watcher. In order to create an assembly and to perform simulations, these are the most common objects that will be created. Figure 2.5 shows a minimal implementation of a sensor. The sensor chooses to stay where it is initially deployed each turn. A more useful implementation might examine the status of the area near where it is located and make a decision about where it might want to move, messages to send, or update internal values to which it can refer in the future.

Figure 2.6 shows a minimal implementation of a deployer that moves every inhabitant to a new random location within the simulation environment. It does this by making a list of inhabitants known to the World object as well as a list of the tiles in the current environment. Then for each inhabitant it picks a random tile and moves the inhabitant to a random point within the chosen tile. While a more useful implementation relative to the sample sensor, this deployer provides a similar function to the Random deployer that is already provided.

```
public class SampleSensor : Inhabitant
{
    public SampleSensor()
        : base()
    {
    }

    public override void Tick()
    {
        base.Tick();

        Velocity = PointF.Empty;
        Action = World.Actions.actStay;
    }

    public override string Info()
    {
        return "A sample sensor object.";
    }
}
```

Figure 2.5. Sample code for a sensor object.

```
public class SampleDeployer : Deployer
{
    public SampleDeployer(World w) : base(w) {}

    public override void Deploy()
    {
        var tiles = new List<Tile>();
        var inhabitants = new List<Inhabitant>();
        foreach (Tile t in World.Tiles.AllTiles)
            tiles.Add(t);
        foreach (Inhabitant i in World.Objects(typeof(Inhabitant)))
            inhabitants.Add(i);
        foreach (Inhabitant i in inhabitants)
        {
            Tile to = tiles[World.Random.Next(tiles.Count - 1)];
            i.Parent.RemoveObject(i);
            i.Position = to.RandomPoint(World.Random);
            to.AddObject(i);
        }
    }

    public override string Info()
    {
        return "This is a sample deployer implementation.";
    }
}
```

Figure 2.6. Sample code for a deployer object.

The source code shown in Figure 2.7 illustrates a minimal implementation of a watcher class. This class subscribes to two of the events that can be thrown during a simulation: the post tick event and the post experiment (step) event. Again, this implementation isn't very useful since it writes a single message to a collection after each turn, and then clears the list after each experiment. However, it illustrates how to subscribe to world events. A common extension would be to add code to save the messages to a log file following each experiment or test run, and to include more useful information in the log message, such as how many tiles contain at least one inhabitant.

```csharp
public class SampleWatcher : Watcher
{
    public SampleWatcher(World world)
        : base(world)
    {
        Log = new StringWriter();
        world.PostTickEvent += new World.PostTickDelegate(OnPostTickEvent);
        world.PostStepEvent += new World.PostStepDelegate(OnPostStepEvent);
    }

    private void OnPostTickEvent(object sender, World.PostTickEventArgs e)
    {
        World.Objects(typeof(SampleSensor));
        Log.WriteLine("Just completed a tick of the clock.");
    }

    private void OnPostStepEvent(object sender, World.PostStepEventArgs e)
    {
        WriteLogToFile();
        Log.Flush();
    }

    public override void RemoveEventHandlers()
    {
        World.PostTickEvent -= this.OnPostTickEvent;
        World.PostStepEvent -= this.OnPostStepEvent;
    }

    private StringWriter Log { get; set; }
    private void WriteLogToFile()
    {
    }
}
```

Figure 2.7. Sample code for a watcher object.

Another class that can be extended is the Incident, which would allow custom behavior to be added to the object that is designed to be the target of certain sensor objects. For example, some implementations allow the incidents to move themselves, or appear/disappear at random (or scheduled) times during the simulation.

Even the Tile object is extensible. Out of the box, rectangle and hexagonal tiles are supported, but other classes (such as triangle or other forms) can be developed.

3. CONCLUSIONS

As has been shown, mobility adds an interesting dimension to the ability of mobile sensor networks to achieve and maintain the system goals. We can draw a number of conclusions from the summation of the problems that have been explored. First, the study of mobile sensor coverage is more complex in light of the constraints imposed by area to be searched, number (density) of sensors, and limited mobility of various forms. Second, although there are a wide array of approaches available when solving problems using mobility, algorithm selection and algorithm design play key roles. Choosing an algorithm carefully, we can successfully mitigate the effects of the limited mobility constraint on coverage using a mobile sensor network. Third, there is much left to explore in this rich field of study.

When considering such algorithms, such as the area traversal routing scheme described in Paper 1 (and illustrated in Figure 2 on Page 19), we see that performance (in terms of coverage of the area and event detection delay) is dependent upon the number of sensors, sensor velocity, and the frequency and duration of events.

In Paper 2 (and illustrated in Figure 1 on Page 40) we showed a taxonomy of coverage problem instances as defined by the relationship between coverage area, number of sensors, and mobility of sensors. There are problem instances for which there is no 1-coverage solutions possible, so we try to maximize coverage. Second, there are instances where coverage is possible, but we attempt to assure it is achieved or try to minimize other metrics, such as time. Finally, there are instances where the number of sensors is ample to assure coverage, in which case we try to show n-coverage, maximize network lifetime, or minimize other metrics such as energy expended over time.

In Paper 3 we showed that disabled mobility affects coverage algorithms in interesting ways and degrades performance of those algorithms as sensors begin to fail. Our results show that deployments that result in an initially well-distributed configuration, and algorithms that spread sensors quickly within the coverage area, are affected less by mobility becoming disabled. We showed a simple mitigation strategy for one algorithm (WGB) that produced demonstrable improvement by altering the heuristic to avoid having sensors send misleading intentions about their mobility plans.

BIBLIOGRAPHY

[1] J-D Nicoud and J-C Zufferey. Toward indoor flying robots. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 787–792. IEEE, 2002.

[2] P. McKerrow. Modelling the draganflyer four-rotor helicopter. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 4, pages 3596–3601 Vol.4, 2004.

[3] Karthik Dantu, Mohammad Rahimi, Hardik Shah, Sandeep Babel, Amit Dhariwal, and Gaurav S. Sukhatme. Robomote: enabling mobility in sensor networks. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 55, Piscataway, NJ, USA, 2005. IEEE Press.

[4] K-Team. Khepera-ii user manual, 2002. `http://ftp.k-team.com/khepera/documentation/Kh2UserManual.pdf` [Online; accessed 15-March-2014].

[5] Amanda Prorok, Adrian Arfire, Alexander Bahr, John R Farserotu, and Alcherio Martinoli. Indoor navigation research with the khepera iii mobile robot: An experimental baseline with a case-study on ultra-wideband positioning. In *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2010.

[6] Dimitrios Lymberopoulos and Andreas Savvides. Xyz: a motion-enabled, power aware sensor node platform for distributed sensor network applications. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 63, Piscataway, NJ, USA, 2005. IEEE Press.

[7] Sriram Chellappan, Xiaole Bai, Bin Ma, and Dong Xuan. Sensor networks deployment using flip-based sensors. In *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[8] Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43(3):19:1–19:51, April 2011.

[9] Douglas W Gage. Command control for many-robot systems. Technical report, DTIC Document, 1992.

[10] G Wang, G. Cao, and T. La Porta. Movement-assisted sensor deployment. In *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[11] Jennifer Casper and Robin R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(3):367–385, 2003.

[12] Yinian Mao and Min Wu. Coordinated sensor deployment for improving secure communications and sensing coverage. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 117–128, New York, NY, USA, 2005. ACM.

[13] Benyuan Liu, Peter Brass, Olivier Dousse, Philippe Nain, and Don Towsley. Mobility improves coverage of sensor networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 300–308, New York, NY, USA, 2005. ACM.

[14] Sriram Chellappan, Xiaole Bai, Bin Ma, and Changqing Xu. Mobility limited flip-based sensor networks deployment. *IEEE Trans. Parallel Distrib. Syst.*, 18(2):199–211, 2007. Member-Xuan,, Dong.

[15] Sriram Chellappan, Wenjun Gu, Xiaole Bai, Dong Xuan, Bin Ma, and Kaizhong Zhang. Deploying wireless sensor networks under limited mobility constraints. *IEEE Transactions on Mobile Computing*, 6(10):1142–1157, 2007.

[16] Wei Wang, Vikram Srinivasan, and Kee Chaing Chua. Trade-offs between mobility and density for coverage in wireless sensor networks. In *MOBICOM*, pages 39–50, 2007.

[17] Zack J. Butler and Daniela Rus. Controlling mobile sensors for monitoring events with coverage constraints. In *ICRA*, pages 1568–1573, 2004.

[18] Guiling Wang, Guohong Cao, and Tom La Porta. A bidding protocol for deploying mobile sensors. In *ICNP '03: Proceedings of the 11th IEEE International Conference on Network Protocols*, page 315, Washington, DC, USA, 2003. IEEE Computer Society.

[19] You-Chiun Wang, Wen-Chih Peng, Min-Hsien Chang, and Yu-Chee Tseng. Exploring load-balance to dispatch mobile sensors in wireless sensor networks. In *ICCCN*, pages 669–674, 2007.

[20] Nabhendra Bisnik, Alhussein Abouzeid, and Volkan Isler. Stochastic event capture using mobile sensors subject to a quality metric. In *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 98–109, New York, NY, USA, 2006. ACM.

[21] Richard Kershner. The number of circles covering a set. *Am. J. Math.*, 61:665–671, 1939.

[22] E L Lawler, J K Lenstra, and A H. The traveling salesman problem, 1985.

[23] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[24] Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward tsp, 2003.

[25] G.N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. In *Foundations of Computer Science, 1976., 17th Annual Symposium on*, pages 216–227, Oct.

[26] Julien Bramel and David Simchi-Levi. Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. *Operations Research*, 44(3):501–509, 1996.

[27] Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006.

[28] T. K. Ralphs, W. R. Pulleyblank, and L. E. Trotter Jr. On capacitated vehicle routing, 1998.

[29] Chandra Chekuri, Nitish Korula, and Martin Pal. Improved algorithms for orienteering and related problems, 2007.

[30] Andrew Howard, Maja J Mataric, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems (DARS02)*, pages 299–308. Citeseer, 2002.

[31] Sasa Slijepcevic and Miodrag Potkonjak. Power efficient organization of wireless sensor networks. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 2, pages 472–476. IEEE, 2001.

[32] J.-W. Lee, Byoung-Suk Choi, and Ju-Jang Lee. Energy-efficient coverage of wireless sensor networks using ant colony optimization with three types of pheromones. *Industrial Informatics, IEEE Transactions on*, 7(3):419–427, 2011.

[33] Junzhao Du, Yawei Li, Hui Liu, and Kewei Sha. On sweep coverage with minimum mobile sensors. In *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, pages 283–290. IEEE, 2010.

[34] Mo Li, Weifang Cheng, Kebin Liu, Yuan He, Xiangyang Li, and Xiangke Liao. Sweep coverage with mobile sensors. *Mobile Computing, IEEE Transactions on*, 10(11):1534–1545, 2011.

[35] Wesley H. Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 27–32, 2001.

[36] Chaomin Luo and S.X. Yang. A real-time cooperative sweeping strategy for multiple cleaning robots. In *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, pages 660–665, 2002.

[37] Richard Kershner. The number of circles covering a set. *American Journal of Mathematics*, 61(3):665–671, 1939.

[38] S Arora. Polynomial-time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, (45):753, 1998.

[39] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[40] Daniel Bienstock, MichelX. Goemans, David Simchi-Levi, and David Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

[41] Austin Armbruster, Michael Gosnell, Bruce McMillin, Mariesa Crow, and M. Crow. Power transmission control using distributed max-flow. In *Proceedings of the 29 th International Computers, Software, and Applications Conference*, pages 256–263, 2005.

[42] K.C.-Y. Chin, S.M. Buhari, and Wee-Hong Ong. Impact of lego sensors in remote controlled robot. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 1777–1782, Feb.

[43] Jennifer Carlson and Robin R Murphy. Reliability analysis of mobile robots. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 274–281. IEEE, 2003.

[44] G Hoblos, M Staroswiecki, and A Aitouche. Optimal design of fault tolerant sensor networks. In *Control Applications, 2000. Proceedings of the 2000 IEEE International Conference on*, pages 467–472. IEEE, 2000.

[45] Prithwish Basu and Jason Redi. Movement control algorithms for realization of fault-tolerant ad hoc robot networks. *Network, IEEE*, 18(4):36–44, 2004.

[46] Monica L Visinsky, Joseph R Cavallaro, and Ian D Walker. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering & System Safety*, 46(2):139–158, 1994.

[47] Mihaela Cardei and Ding-Zhu Du. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, 11(3):333–340, 2005.

[48] Yunxia Chen and Qing Zhao. On the lifetime of wireless sensor networks. *Communications Letters, IEEE*, 9(11):976–978, 2005.

[49] Yunxia Chen, Qing Zhao, Vikram Krishnamurthy, and Dejan Djonin. Transmission scheduling for optimizing sensor network lifetime: A stochastic shortest path approach. *Signal Processing, IEEE Transactions on*, 55(5):2294–2309, 2007.

[50] Dan Wang, Jiangchuan Liu, and Qian Zhang. Probabilistic field coverage using a hybrid network of static and mobile sensors. In *Quality of Service, 2007 Fifteenth IEEE International Workshop on*, pages 56–64. IEEE, 2007.

[51] Isabel Dietrich and Falko Dressler. On the lifetime of wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 5(1):5, 2009.

[52] Jennifer Carlson and Robin R Murphy. How ugvs physically fail in the field. *Robotics, IEEE Transactions on*, 21(3):423–437, 2005.

[53] C.D. Lai, M. Xie, and D.N.P. Murthy. Ch. 3. bathtub-shaped failure rate life distributions. In N. Balakrishnan and C.R. Rao, editors, *Advances in Reliability*, volume 20 of *Handbook of Statistics*, pages 69 – 104. Elsevier, 2001.

[54] Joseph R Cavallaro and Ian D Walker. A survey of nasa and military standards on fault tolerance and reliability applied to robotics. In *NASA CONFERENCE PUBLICATION*, pages 282–282. NASA, 1994.

[55] J. Carlson, R.R. Murphy, and A. Nelson. Follow-up analysis of mobile robot failures. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4987–4994 Vol.5, 2004.

[56] Stephen Stancliff, John M Dolan, and Ashitey Trebi-Ollennu. Towards a predictive model of robot reliability. 2005.

[57] Emre Atsan and Öznur Özkasap. A classification and performance comparison of mobility models for ad hoc networks. In *Ad-Hoc, Mobile, and Wireless Networks*, pages 444–457. Springer, 2006.

[58] Guiling Wang, Guohong Cao, and Tom La Porta. Proxy-based sensor deployment for mobile sensor networks. In *Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on*, pages 493–502. IEEE, 2004.

[59] Mark Snyder, Sriram Chellappan, and Mayur Thakur. Exploratory coverage in limited mobility sensor networks. In *Network-Based Information Systems, 2013. NBIS'13. International Conference on*. IEEE, September 2013.

[60] Yi Zou and Krishnendu Chakrabarty. Sensor deployment and target localization based on virtual forces. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1293–1303. IEEE, 2003.

[61] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[62] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: associations between software development and crowdsourced knowledge. In *Social Computing (SocialCom), 2013 International Conference on*, pages 188–195. IEEE, 2013.

[63] Kenneth A De Jong. *Evolutionary computation: a unified approach*, volume 262041944. MIT press Cambridge, 2006.

[64] Kenneth Alan De Jong. Analysis of the behavior of a class of genetic adaptive systems. 1975.

VITA

Mark Edward Snyder was born May 5, 1965 in Independence, Missouri, USA. Snyder's passion for computers began after he was introduced to the TRS-80 and began programming in BASIC in 1978. He received a secondary degree and vocational computer science training from Fort Osage High School in May, 1983, followed by a Bachelor of Science degree in Computer Science from Central Missouri State University at Warrensburg, Missouri, in May, 1987.

Snyder's career leaned toward the travel industry early on, starting with computerized reservation software development for numerous airline companies, including work with databases, services, middleware, and presentation tier application development projects across a diverse landscape of programming languages, application programming interfaces, frameworks, protocols, and systems.

While at Microsoft, Snyder worked in the Consumer Division and Microsoft Research. He was part of a team that invented the online travel agency web site Expedia.com, and then a team that created Allegiance, a massively-multiplayer, space combat simulation game.

After two decades in the commercial software industry, Snyder turned his passion for computer science to research, earning a Master of Science in Computer Science from the Missouri University of Science and Technology at Rolla, Missouri, in December, 2008. In 2010, Snyder returned once again to join Expedia and currently manages a software development team. In May, 2014, he received his Ph.D. in Computer Science from Missouri University of Science and Technology.

When not working, Snyder reads science fiction novels, watches quirky television series reruns, travels with his wife, and enjoys the outdoors with his father and sons. At other times in his life, he has been a cattle farmer, fireman, and teacher.