

CELLULAR AUTOMATON BASED ALGORITHMS FOR
WIRELESS SENSOR NETWORKS

by

SALIMUR CHOUDHURY

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada

November 2012

Copyright © Salimur Choudhury, 2012

Abstract

Wireless sensor networks have been used in different applications due to the advancement of sensor technology. These uses also have raised different optimization issues. Most of the algorithms proposed as solutions to the various optimization problems are either centralized or distributed which are not ideal for these real life applications. Very few strictly local algorithms for wireless sensor networks exist in the literature. In this thesis, we consider some of these optimization problems of sensor networks, for example, sleep-wake scheduling, mobile dispersion, mobile object monitoring, and gathering problems. We also consider the depth adjustment problem of underwater sensor networks. We design cellular automaton based local algorithms for these problems. The cellular automaton is a bioinspired model used to model different physical systems including wireless sensor networks. One of the main advantages of using cellular automaton based algorithms is that they need very little local information to compute a solution. We perform different simulations and analysis and find that our algorithms are efficient in practice.

Acknowledgments

I express my deep acknowledgment and profound sense of gratitude to my supervisors Dr. Selim G. Akl and Dr. Kai Salomaa for their inspiring guidance, helpful suggestions and persistent encouragement as well as close and constant supervisions throughout the period of my Ph.D. study. I specially thank Professors Ivan Stojmenovic, Hossam Hassanein, Bob Crawford, and Thomas Dean for their comments and suggestions. I am very much thankful to my family and fellow graduate students of the department and all the friends at Kingston for their continuous encouragement that helped me to complete this thesis. A special thank to my wife Shofia Akhtar for being with me always and without her support it would be very much difficult to complete this thesis.

I thank Queen's University, School of Computing, Ontario Ministry of Training, Colleges and Universities, and my supervisors for the financial assistantships.

Dedication

I dedicate this thesis to my family.

Acronyms

- WSN : Wireless Sensor Network
- MWSN : Mobile Wireless Sensor Network
- UWSN : Underwater Sensor Network
- CA : Cellular Automata
- R_s : Sensing Radius of a Sensor.
- R_c : Communication Radius of a Sensor.
- m_i : i^{th} mobile sensor.
- aCOV: Average Coverage of a Network.
- aHD: Average Hop Distance of the Network.
- aSCC: Strongly Connected Coverage of a Network
- raSCC: Ratio of the Average Strongly Connected Coverage.
- AVGD: Average number of movements by a sensor.
- MAXD: The maximum number of movements by a sensor.

Co-Authorship

- S. Choudhury, K. Salomaa and S. G. Akl, “Cellular automaton based algorithms for object monitoring problem of wireless sensor networks”, submitted for publication.
- S. Choudhury, K. Salomaa and S. G. Akl, “Cellular Automaton Based Algorithms for the Dispersion of Mobile Wireless Sensor Networks”, submitted for publication.
- S. Choudhury, K. Salomaa and S. G. Akl, “A Cellular Automaton Model for Wireless Sensor Networks”, *Journal of Cellular Automata*, Volume 7, Issue 3, 2012.
- S. Choudhury, K. Salomaa and S. G. Akl, “Cellular Automaton Based Algorithms for Depth Adjustment in Underwater Sensor Networks”, *Proceedings of the Sixth International Workshop on Wireless Sensor, Actuator and Robot Networks*, Las Vegas, October 2012.
- S. Choudhury, K. Salomaa and S. G. Akl, “Cellular Automaton Based Motion Planning Algorithms for Mobile Sensor Networks”, *Proceedings of the First International Conference on the Theory and Practice of Natural Computing*,

Tarragona, Spain, October, 2012.

- S. Choudhury, K. Salomaa and S. G. Akl, “A Cellular Automaton Model for Connectivity Preserving Deployment of Mobile Wireless Sensors”, to appear in 2nd IEEE International Workshop on Smart Communication Protocols and Algorithms (ICC’12 WS - SCPA), Ottawa, Canada, June 10-15, 2012.
- S. Choudhury, S. G. Akl and K. Salomaa, “Energy Efficient Cellular Automaton Based Algorithms for Mobile Wireless Sensor Networks”, In Proceedings of the 2012 IEEE Wireless Communications and Networking Conference (WCNC 2012), Paris, France, April 1-4, 2012, pp. 2341-2346
- S. Choudhury, K. Salomaa and S. G. Akl, “A Cellular Automaton Model for Wireless Sensor Networks”, Proceedings of the Twenty Second IASTED International Symposium on Modelling and Simulation (MS), Calgary, Alberta, Canada, July 4-6, 2011, p.p 190-195.

Statement of Originality

I hereby certify that the research presented in this dissertation is my own, conducted under the supervision of Dr. Selim G. Akl and Dr. Kai Salomaa. Ideas and techniques that are not a product of my own work are cited, or, in cases where citations are not available, are presented using language that indicates they existed prior to this work.

Table of Contents

| | |
|---|----------|
| Abstract | i |
| Acknowledgments | ii |
| Dedication | iii |
| Acronyms | iv |
| Co-Authorship | v |
| Statement of Originality | vii |
| Table of Contents | viii |
| List of Tables | xii |
| List of Figures | xiv |
| Chapter 1: | |
| Introduction | 1 |
| 1.1 Challenges | 3 |
| 1.2 Contributions of the Thesis | 9 |
| 1.3 Organization of the Thesis | 14 |

Chapter 2:

| | |
|---|-----------|
| Preliminaries | 16 |
| 2.1 Wireless Sensor Networks | 16 |
| 2.2 Mobile Wireless Sensor Networks | 24 |
| 2.3 Underwater Sensor Networks | 27 |
| 2.4 Cellular Automata | 30 |
| 2.5 Confidence Interval | 36 |

Chapter 3:

| | |
|--|-----------|
| Sleep-Wake Scheduling of Sensors | 38 |
| 3.1 Introduction | 38 |
| 3.2 Representing a Wireless Sensor Network as a Cellular Automaton . . | 40 |
| 3.3 The Coverage and the Object Detection Problems | 42 |
| 3.4 Simulation and Results | 43 |
| 3.5 Summary | 56 |

Chapter 4:

| | |
|--|-----------|
| Autonomous Deployment of Mobile Sensors | 59 |
| 4.1 Introduction | 59 |
| 4.2 Related Work | 61 |
| 4.3 System Model of Mobile Sensor Networks | 63 |
| 4.4 Algorithms for Maximizing Coverage | 63 |
| 4.5 Connectivity Preserving Dispersion Problem | 78 |
| 4.6 Summary | 98 |

Chapter 5:

| | |
|--|------------|
| Synchronous Gathering of Mobile Sensors | 101 |
| 5.1 Introduction | 101 |
| 5.2 Problem Statement | 102 |
| 5.3 Algorithm | 103 |
| 5.4 Analysis | 105 |
| 5.5 Simulations and Results | 106 |
| 5.6 Summary | 112 |

Chapter 6:

| | |
|--|------------|
| Minimizing Chain Length Problem | 114 |
| 6.1 Introduction | 114 |
| 6.2 Problem Definition | 115 |
| 6.3 Algorithms | 116 |
| 6.4 Simulations and Results | 120 |
| 6.5 Summary | 123 |

Chapter 7:

| | |
|---|------------|
| Object Monitoring in MWSN | 125 |
| 7.1 Problem Definition and Algorithms | 126 |
| 7.2 Simulation and Results | 130 |
| 7.3 Summary | 135 |

Chapter 8:

| | |
|--|------------|
| Underwater Mobile Sensor Networks | 136 |
| 8.1 Introduction | 136 |

| | | |
|-------------------|---|------------|
| 8.2 | Problem Definition and System Model | 138 |
| 8.3 | Algorithms | 139 |
| 8.4 | Simulations and Results | 144 |
| 8.5 | Summary | 149 |
| Chapter 9: | | |
| | Summary and Conclusions | 151 |
| 9.1 | Summary | 151 |
| 9.2 | Future Work | 154 |
| | Bibliography | 157 |

List of Tables

| | | |
|-----|---|-----|
| 3.1 | Area Under the Curves of Figure 3.3 | 46 |
| 3.2 | Area Under the Curves of Figure 3.8 | 51 |
| 3.3 | Experimental Results on Object Detection Problem (Experiment 1) . | 54 |
| 3.4 | Experimental Results on Object Detection Problem (Experiment 2) . | 56 |
| 4.1 | Experimental Results on Different Upper Bounds on the Movements in Scenario 1 in the Case of $R_c = 4$ and $R_s = 1$ | 71 |
| 4.2 | Experimental Results on Different Upper Bounds on the Movements in Scenario 2 in the Case of $R_c = 4$ and $R_s = 1$ | 75 |
| 4.3 | Experimental Results on Different Upper Bounds on the Movements in Scenario 3 in the case of $R_c = 4$ and $R_s = 1$ | 78 |
| 4.4 | Comparison Results for Different Multiplication Factors with 180°- moveback rule | 92 |
| 6.1 | Comparison of Algorithms for Different Scenarios with Different Num- bers of Sensors | 124 |
| 7.1 | Experimental Results on Object Monitoring Problem with 100 mobile objects and $R_c = 3$ | 134 |

| | | |
|-----|--|-----|
| 7.2 | Experimental Results on Object Monitoring Problem with 100 mobile objects and $R_c = 2$ | 135 |
|-----|--|-----|

List of Figures

| | | |
|------|---|----|
| 2.1 | The Components of a Sensor | 17 |
| 2.2 | Multihop Routing | 19 |
| 2.3 | Architecture of a Two Dimensional Underwater Sensor Networks [4] . | 29 |
| 2.4 | Architecture of a Three Dimensional Underwater Sensor Networks [4] | 30 |
| 2.5 | (a) Von Neumann Neighborhood, (b) Moore Neighborhood | 31 |
| 2.6 | Moore Neighborhood with Radius-2 | 31 |
| 3.1 | Comparison of the Radius 1 Rules of the Coverage Problem | 47 |
| 3.2 | Comparison of the Radius 2 Rules of the Coverage Problem | 48 |
| 3.3 | Comparison of the Rules of the Coverage Problem | 48 |
| 3.4 | Number of Alive Sensors as a Function of Time for the Best Two Rules of the Algorithms | 49 |
| 3.5 | The Residual Energy of the Network as a Function of Time | 49 |
| 3.6 | Comparison of the Rules of Radius 1 Neighborhood Without the Counter | 50 |
| 3.7 | Comparison of the Rules of Radius 2 Neighborhood Without the Counter | 50 |
| 3.8 | Comparison of the Rules Without the Counter | 51 |
| 3.9 | A Comparison of Radius 1 Rules on Object Detection (Experiment 1) | 54 |
| 3.10 | A Comparison of Radius 2 Rules on Object Detection (Experiment 1) | 54 |

| | | |
|------|---|----|
| 3.11 | A Comparison of Radius 1(1/1) and Radius 2(1/1) Algorithms on Object Detection (Experiment 1) | 55 |
| 3.12 | A Comparison of Radius 1 Rules on Object Detection (Experiment 2) | 56 |
| 3.13 | A Comparison of Radius 2 Rules on Object Detection (Experiment 2) | 57 |
| 3.14 | A Comparison of Radius 1(1/1) and Radius 2(1/1) Algorithms on Object Detection (Experiment 2) | 57 |
| 4.1 | An Example of Mobile Wireless Sensor Network | 64 |
| 4.2 | The Four Quadrants of a Cell (i, j) When R_c is 4 | 65 |
| 4.3 | Scenario 1 | 68 |
| 4.4 | Scenario 1: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 4$ and $R_s = 1$ | 69 |
| 4.5 | Scenario 1: Total Number of Moves of the Sensors for Different Rules in the Case of $R_c = 4$ and $R_s = 1$ | 69 |
| 4.6 | Scenario 1: The Positions of the Sensors at Time 100 (3/3) in the Case of $R_c = 4$ and $R_s = 1$ for an Experiment | 70 |
| 4.7 | Scenario 1: The Positions of the Sensors at Time 100 (COUNT) in the Case of $R_c = 4$ and $R_s = 1$ for an Experiment | 70 |
| 4.8 | Scenario 1: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 3$ and $R_s = 1$ | 72 |
| 4.9 | Scenario 1: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 2$ and $R_s = 1$ | 72 |
| 4.10 | Scenario 2 | 73 |
| 4.11 | Scenario 2: Comparison of the Total Coverage for Different Rules . . . | 73 |
| 4.12 | Scenario 2: Total Number of Moves of the Sensors for Different Rules | 74 |

| | |
|---|----|
| 4.13 Scenario 2: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 3$ and $R_s = 1$ | 74 |
| 4.14 Scenario 2: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 2$ and $R_s = 1$ | 74 |
| 4.15 Scenario 3 | 76 |
| 4.16 Scenario 3: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 4$ and $R_s = 1$ | 76 |
| 4.17 Scenario 3: Total Number of Moves of the Sensors for Different Rules in the Case of $R_c = 4$ and $R_s = 1$ | 76 |
| 4.18 Scenario 3: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 3$ and $R_s = 1$ | 77 |
| 4.19 Scenario 3: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 2$ and $R_s = 1$ | 77 |
| 4.20 Placements of Total 100 Sensors for $R_c = 3$ | 79 |
| 4.21 An Example of a Cycle and the Usefulness of Multiplication Factor | 83 |
| 4.22 Example: Connectivity Breaks. $R_c = 3$ | 85 |
| 4.23 Example: Additional Blocking Rule. $R_c = 3$ | 85 |
| 4.24 A situation Where Sensor m_1 in C_1 cannot Move in the Positive x (or y) Direction | 86 |
| 4.25 Neighbors Break Connectivity for $R_c = 3$ | 86 |
| 4.26 Quadrant Move Back Rule with $R_c = 3$ | 87 |
| 4.27 Move Back Rules for a Cell with Multiple Sensors with $R_c = 3$. (a) Quadrant Move Back, (b) 180°-move back | 87 |
| 4.28 A Random Initial Configuration for 1000 sensors | 89 |

| | | |
|------|--|-----|
| 4.29 | Star Benchmark with $R_c = 3$ | 90 |
| 4.30 | Example of a big hole | 91 |
| 4.31 | Comparison of the Final Configurations: (a) Probabilistic Quadrant Move Back, (b) Probabilistic 180°-move back rule | 94 |
| 4.32 | Example of a Hole in Quadrant Move Back | 94 |
| 4.33 | Comparison of aCOV Among Two Probabilistic Algorithms for Differ- ent Number of Mobile Sensors With $R_c = 3$ and $R_s = 2$ | 95 |
| 4.34 | Comparison of aSCC Among Two Probabilistic Algorithms for Differ- ent Number of Mobile Sensors With $R_c = 3$ and $R_s = 2$ | 95 |
| 4.35 | Comparison of aSCC for Different Number of Mobile Sensors with $R_c =$ 2 and $R_s = 1$ | 95 |
| 4.36 | Comparison of aSCC for Different Number of Mobile Sensors with $R_c =$ 3 and $R_s = 1$ | 96 |
| 4.37 | Example: Final Configurations With 1000 Sensors for a) (3,2) and b) (2,1) | 96 |
| 4.38 | Average Coverage Vs Time period (Probabilistic Quadrant Move Back) | 97 |
| 4.39 | Average Coverage Vs Time period (Probabilistic 180°-move back) . . | 98 |
| 4.40 | raSCC With Respect to Square Benchmark for the Probabilistic with Quadrant Move Back Algorithm for Different (R_c, R_s) Pairs | 99 |
| 4.41 | raSCC With Respect to Star Benchmark for the Probabilistic With Quadrant Move Back Algorithm for Different (R_c, R_s) pairs | 99 |
| 5.1 | An Example of the Final Step of the Algorithm | 105 |
| 5.2 | An Example Where Connectivity Breaks | 107 |
| 5.3 | A Square Configuration | 109 |

| | | |
|------|---|-----|
| 5.4 | Comparison of Time Periods Taken by the Algorithms | 110 |
| 5.5 | Comparison of AVGD | 110 |
| 5.6 | Comparison of MAXD | 111 |
| 5.7 | An Example of a Random Configuration of the Mobile Sensors with $R_c = 2$ | 111 |
| 5.8 | Comparison of Time Periods Taken by the Algorithms | 112 |
| 5.9 | Comparison of AVGD | 112 |
| 5.10 | Comparison of MAXD | 113 |
| 6.1 | Scenario 1: A Chain Configuration | 116 |
| 6.2 | An Example of a Cycle | 119 |
| 6.3 | Scenario 2: A Chain Configuration | 121 |
| 6.4 | Scenario 3: A Chain Configuration | 121 |
| 6.5 | Scenario 4: A Chain Configuration | 122 |
| 6.6 | An Example of a Simulation: (a) Initial Configuration, (b) Configu- ration after 5 Time periods, (c) Configuration after 20 Time periods, and (d) Final Configuration | 122 |
| 7.1 | An Example of of Object Monitoring Problem with $R_c = 2$ and $R_s = 1$ | 130 |
| 7.2 | Comparison Results for 200 Mobile Sensors and 100 Mobile Objects with $R_c = 3$ and with 100×100 Grid | 132 |
| 7.3 | Comparison Results for 300 Mobile Sensors and 100 Mobile Objects with $R_c = 3$ and with 100×100 Grid | 133 |
| 8.1 | An Example of Underwater Networks | 137 |
| 8.2 | Blocking Rule with $R_c = 3$ | 141 |

| | | |
|------|--|-----|
| 8.3 | A Blocking Rule (half plane) with $R_c = 3$ | 142 |
| 8.4 | Extended Blocking Rule at 2D with $R_c = 3$ | 143 |
| 8.5 | Second Type of Deployment with $R_c = 3$ | 145 |
| 8.6 | Final Positions of Total 75 Sensors with $R_c = 3$ and $R_s = 1$ When Sensors Initially the Sensors are Deployed at all Surface Points. | 146 |
| 8.7 | Final Positions of Total 75 Sensors with $R_c = 3$ and $R_s = 1$ at Some Surface cells. Each Cells are at Distance 3 from the Neighboring Cell. | 147 |
| 8.8 | Final Positions of the Sensors (300) for $R_c = 3$ | 148 |
| 8.9 | Coverage with Time for 300 Sensors with $R_c = 3$ | 148 |
| 8.10 | Benchmark Configuration with $R_c = 3$ | 149 |
| 8.11 | Final Positions of the Sensors with Extended Blocking Rules for $R_c = 3$ | 149 |
| 8.12 | Strongly Connected Coverage Produced by the Algorithm (as a Per- centage of the Benchmark) When Initially the Sensors are Deployed at All Surface Points. | 150 |
| 8.13 | Strongly Connected Coverage Produced by the Algorithm (as a Per- centage of the Benchmark) When Initially the Sensors are Deployed at Some Surface Cells. Each Cells are at Distance R_c from the Neighbor- ing Cells | 150 |

Chapter 1

Introduction

Due to the advancement of sensor communication technology, wireless sensor networks have been used in many applications. The sensors are the main components of a *wireless sensor network*. A *sensor* is a very low cost small device that has limited battery power, short communication range, limited processing power and limited memory. A wireless sensor network (WSN) forms a distributed information processing system that gathers and processes different attributes of the network, for example, humidity, temperature, etc. A traditional wireless sensor network also includes single or multiple base stations that gather data from the sensors [118].

Each sensor of a sensor network has a sensing radius and a communication radius. A sensor can sense or monitor the region that falls within its sensing radius and communicates with other sensors that are within its communication radius. The sensors with whom a sensor can communicate are called the neighbors of the sensor. A typical wireless sensor network consists of hundreds, or even thousands, of sensors. These sensors are deployed in the monitored area and typically centralized controls are absent on these sensors [72]. These nodes are also unattended due to typical

applications of sensor networks, it is not possible to have a human operator to directly attend to individual sensors. The sensors use each other (multi-hop communication) to route the information that they sense to the base stations for further processing.

Habitat monitoring is an important application of wireless sensor networks. Mainwaring *et.al.* [77] designed a wireless sensor network on Great Duck Island, Maine, USA to monitor the behavior of storm petrel. Some other habitat monitoring applications are considered in [24, 22, 112, 66]. The other important applications of wireless sensor networks are health monitoring systems [96, 78], home applications [100], etc.

Unlike traditional networking technology, energy is one of the major constraints that limits the usability of wireless sensor networks [118]. The energy of a sensor decays with time. A typical sensor can be in one of the two modes, asleep and awake. When a sensor is in the awake mode it can sense its region and also can communicate with its neighboring sensors while in the asleep mode, it can not sense or communicate. However the energy of a sensor decays in both modes. A sensor spends much more energy in the awake state than in the asleep state and the amount of energy needed for the purpose of communication increases with the increase of radius. When a sensor loses all of its energy we consider that sensor as dead. Designing energy aware techniques for different applications of wireless sensor networks is a well studied research area [62].

As the sensors spend more energy in communication and they use multi-hop communication, the sensors that are closer to the base station can be overloaded and can die early. This problem can be solved using mobile sinks [9]. When a component of a wireless sensor network (either a sensor or a sink) has moving capability then we call the network a *Mobile Wireless Sensor Network* (MWSN). Moreover, in different

applications, for example, military field, mobile objects monitoring, etc., the sensors cannot be deployed in their desired positions by a centralized (or human) operator. In these cases, we can deploy the sensors densely and they can autonomously move within the network to improve the performance of the network if they have a moving capability [76] .

In most of the applications of a WSN or a MWSN, two metrics are considered to measure the performance of the network [39]. One is the coverage and the other is the network lifetime. Coverage is defined as the area monitored by the network. The definition of network lifetime varies with the application. In some cases, we define a network lifetime as the time elapsed until a sensor of the network dies, while in some cases network lifetime refers to the time until a given fraction of the sensors die. The other definition of network lifetime is the time elapsed “until all the sensors are dead” [39].

Underwater sensor networks (UWSN) are relatively new applications of wireless sensor networks [4, 87]. In a typical UWSN, sensors are used to monitor different aspects of seas, lakes, etc. The sensors of a UWSN can be either static or mobile. Differing from a land based MWSN, the sensors of a mobile UWSN move in three dimensions.

1.1 Challenges

Deployment is one of the main challenges of the effective use of wireless sensor networks. In a typical deployment, a number of metrics are considered. One of the main metrics is the energy of the sensors. As the energy of the sensors is limited and they decay with time we need to deploy the sensors in a way that enables us to extend

their lifetime as long as possible. There are other metrics often considered at the deployment, for example, connectivity, coverage, etc [106].

Most of the algorithms found in the literature related to the sensor network deployment are either global or distributed [92]. In a real environment it is not feasible to implement a global algorithm. On the other hand, a distributed algorithm needs more communication and message passing which makes the algorithm complex. A local algorithm can be a good candidate solution for this type of optimization problems. Even the local algorithms that have been designed for WSN are quite complex in terms of processing and memory needed [101].

The *Cellular Automaton* is a biologically inspired model that has been widely used to model different physical systems [52, 61, 25]. We can model a 2–dimensional cellular automaton as a 2–dimensional grid. Each cell of the grid is in one of a finite number of states. The cells change state synchronously at discrete time steps and the new state depends always only on states of a local neighborhood.

One of the major benefits of using a cellular automaton based model is that it needs very limited local information to compute the solution. We can also perform extensive simulations if we use cellular automaton models as cellular automata are easy to implement. The main challenge is to design good local rules that give a satisfactory global outcome for different optimization problems. In the following subsections we describe and discuss some optimization problems and related challenges.

1.1.1 Sleep-wake Scheduling of Sensors

Due to the limited energy of the sensors, it is often advantageous to place some of the sensors in an asleep state. Below we describe the sleep-wake scheduling problem

in some more detail.

Each sensor of a WSN can be either in the awake or in the asleep state. If a sensor is in the awake state then the sensor can sense its region of interest and can communicate with its neighbors while if the sensor is in the asleep state then the sensor can not perform any such operation. The energy of a sensor decays in both asleep or awake states. However, it loses much more of its energy when it is in awake state and even in awake state it loses more energy when it communicates with its neighbors. When a sensor loses all of its energy then the sensor is considered as a dead node. A network can not perform any operation when all of its sensors become dead.

Some parts of the network can be over monitored if the region falls within the sensing radius of more than one sensor and some parts can be totally uncovered. Scheduling sensors between asleep and awake states can solve this issue. Moreover, an optimal scheduling can extend the network lifetime.

There are lots of solutions in the literature for such scheduling problems. Most of the solutions are either centralized or distributed [23, 40, 10, 70, 27]. There exist some local algorithms which are quite complex to implement in real applications. Cunha *et.al.* [39] propose cellular automaton based local algorithms that are quite easy to implement and need very little local information by any sensor to be a part of the scheduling algorithm. However, their algorithm suffers some sudden falls of coverage. Improving the algorithm proposed in [39] and designing a cellular automaton based algorithm that increases network lifetime and coverage while solving the sudden falls of coverage are important research questions.

1.1.2 Autonomous Deployment of MWSN

In a MWSN application, sensors can be deployed in a network to cover some region of the network. The coverage of a network is defined as the area that is covered by its sensors. In different applications of mobile sensor networks, it is not possible to deploy the sensors deterministically so that they can maximize the coverage. More commonly, they are deployed randomly and the sensors are required to disperse autonomously using algorithms to maximize the coverage of the network [76]. A major concern for the dispersal algorithms is the preservation of connectivity. Once the sensors try to move by themselves, they can break the connectivity. Connectivity is an important aspect of the network as it is used to route the data among the nodes. We call this problem *the Mobile Dispersion Problem*. Most of the solutions that have been proposed for this problem are either global or distributed algorithms that are also, arguably, highly complex [68, 88, 58, 111, 36].

Torbey [106] proposes a cellular automaton based local algorithm to increase the coverage of a network that is restricted to a bounded area. However, the algorithm does not explicitly consider the connectivity preservation criterion and he does not consider any random initial deployment.

To the best of our knowledge, there is no good strictly local solution for maximizing coverage while maintaining connectivity for any random dense initial deployment.

1.1.3 Synchronous Gathering Problem

After dispersing autonomously, sensors can try to gather in a location for various reasons, for example, exchanging information or to be collected from a single point.

Designing local algorithms for such gathering problem is an important research question in both sensor networks and in robotics [12, 43]. We can view this problem as a reverse problem of the dispersion problem. However, the reverse of the algorithms proposed for the other problem are not applicable in this case. To the best of our knowledge, there is no cellular automaton based local algorithm for the synchronous gathering problem though there exists one local algorithm which is arguably complex [12].

1.1.4 Minimizing Chain length Problem

Often sensors can make a long windy chain in a network, where each sensor except the two end points has exactly two neighbors: left and right [46, 65]. The two end points of the network are fixed and other sensors can move autonomously. There can be a mobile explorer who wants to visit each and every sensor of such a network. If the network is too windy it might take a long time for it to traverse such a network. Minimizing the chain length can help the mobile explorer to visit the networks in minimum time and with minimum energy. Designing a cellular automaton based local algorithm for this problem is an open research issue.

1.1.5 Monitoring Mobile Objects

Monitoring mobile objects is an important application of a MWSN [120, 107]. Initially different mobile objects and sensors can be deployed in a dense area of the network and the objects can be monitored. However, when the objects start moving they can not be monitored if the sensors do not move accordingly. Designing a motion planning algorithm for such a problem is an important research question. While many

algorithms have been proposed for different monitoring problems [103, 111, 63, 107], to the best of our knowledge there is no cellular automaton based local algorithm for this problem.

1.1.6 Underwater Sensor Networks

About seventy percent of our world is covered by water. We can answer a lot of research questions if we can explore under water. Sensor technology has given us that opportunity. An Underwater Wireless Sensor Network (UWSN) consists of sensors that are deployed underwater at various depths and are connected with some sinks and the sinks are connected with surface stations.

Three dimensional wireless sensor networks have become popular due to their applications in underwater sensing [4, 87]. Often algorithmic solutions to analogous problems in two and three dimensions, respectively, can be significantly different. There are some algorithms that have been proposed for different optimization problems for three dimensional wireless sensor networks [16, 60, 7, 6]. One of the important optimization problems is the depth adjustment of the mobile sensors.

In a mobile underwater sensor network, sensors can be deployed at the surface level. The sensors can adjust their depth autonomously to maximize the coverage of the network while maintaining the connectivity with the surface stations. There are very few algorithms proposed for this problem which are not purely local [3, 44, 45]. Designing cellular automaton based local algorithms for the depth adjustment of the underwater sensor networks is an important research issue.

1.2 Contributions of the Thesis

In this thesis we consider cellular automaton based local algorithms for various optimization problems of WSN, MWSN, and UWSN.

In the following subsections we summarize the contributions of this thesis.

1.2.1 Algorithms for Sleep-Wake Scheduling

In this section of the thesis, we design an algorithm for the sleep-wake scheduling problem that is based on information from the radius-2 neighborhood of the sensors (neighbors that are at most distance 2 away from the sensor) to improve the coverage and lifetime of a wireless sensor network. Moreover, we consider two different energy levels for sensing and communicating among neighbors. To make the comparison with the earlier radius 1 algorithm [39] realistic, in our algorithm the sensing radius remains 1 and the larger radius 2 neighborhood is used only for communicating with neighbors. Naturally, the energy used for communication depends on the square of the communication radius, and this is taken into account when calculating the energy use of sensors. We also solve the sudden falls of the coverage occurring in the algorithm of [39] using a probabilistic technique. We develop a CA simulator and compare the different rules to implement the CA model. We compare the algorithms in terms of different metrics, for example, network lifetime, coverage, total number of alive sensors and energy left with the increase of time. We find that our algorithm gives better results compared to the earlier one.

Object detecting in a WSN is one of the well studied research problems [75, 13, 67, 14]. In an animal behavior monitoring environment or in a military field, sensors are usually deployed to detect animals or enemies, respectively. To this end, algorithms

are proposed that increase the probability of success. In such a situation network lifetime is also an important factor. As soon as the network dies, we cannot detect any objects. We use our coverage algorithm for the object detection problem too and compare the results with an earlier algorithm [39] in terms of the number of undetected objects as a function of time.

The results of this section have been published in [32, 33]

1.2.2 Autonomous Deployment of Mobile Wireless Sensor Networks

In this part of the thesis, first we consider an improved CA based algorithm to maximize the coverage of a MWSN. We divide the neighborhood of a sensor into quadrants and then move the sensor to that quadrant that has the minimum number of sensors [30].

In the earlier related work [106], connectivity was not considered to be a major concern and there was no explicit rule for preserving the connectivity. We devise an algorithm that considers connectivity as well as coverage. Moreover, we allow that one cell can contain more than one sensor which is more general than the set-up considered in [106]. We consider the following scenarios for the deployment of sensors:

- Sensors are deployed in one or more square blocks where each cell of a block contains exactly one sensor [31].
- Sensors are initially randomly deployed in a square block where some cells can contain more than one sensor while some cells can contain no sensor [29].

We consider different sensing and communication radii in both cases. Finally, we develop a simulator that compares different algorithms for the dispersion problem. We compare our algorithms with benchmark configurations that result from a deterministic placement of sensors [31, 29]. We compare different metrics, for example, coverage, time taken to reach the satisfactory coverage, average hop distances among the sensors, etc with two benchmarks and find competitive results.

1.2.3 Synchronous Gathering of Mobile Wireless Sensor Networks

We consider *the synchronous gathering problem* of mobile sensors and design a CA based algorithm for it. In this scenario, the sensors are initially dispersed and need to autonomously move to a single location where they can all communicate with each other directly or where they can be collected for later use.

An analogous problem has been studied in robotics [12, 43] under the name of synchronous gathering problem. A local algorithm proposed in [12] is considerably more complex than our CA based algorithm. To the best of our knowledge, we propose the first CA based algorithm for the synchronous gathering problem.

We consider the following scenarios for the problem:

- Initially sensors are deployed as a square configuration where the distance between adjacent sensors is the communication radius.
- The initial configuration is produced by the algorithm developed for the mobile dispersion problem (that was discussed in section 1.2.2). While such distributions are not random in a precise sense, they can be considered as reasonably

“random” configurations where the sensors are connected.

We consider different communication radii for this problem and develop a simulator to compare the algorithms in terms of time and the amount of movement it takes to reach the final configuration. We prove that using our algorithm sensors always converge in a finite amount of time. We compare our algorithm with a local algorithm proposed in [49] and find that our algorithm takes less time than their algorithm to converge.

1.2.4 Minimizing Chain Length Problem

In this part of the thesis we propose cellular automaton based algorithms for the minimizing chain length problem. We consider different types of initial windy configurations of the sensors and design one probabilistic and one deterministic algorithm to minimize the chain length.

We consider two different metrics, the *height* and the *length* of the chain to compare our algorithms. The height of a configuration is the maximum distance between a mobile sensor and a shortest line connecting two end points at a given time while the length of a configuration is the sum of distances between designated neighbors along the line of sensors from one end point to another at a given time. We find that both algorithms give the optimal solution for different initial configurations in practice while the deterministic algorithm takes less time than the probabilistic algorithm. To the best of our knowledge, these are the first cellular automaton based algorithms for this problem.

1.2.5 Mobile Objects Monitoring in a Mobile Wireless Sensor Network

We design a cellular automaton based algorithm for monitoring mobile objects in a MWSN restricted to a bounded area. To the best of our knowledge this is the first cellular automaton based algorithm for this problem. The mobile objects and the mobile sensors are initially deployed in a square block where all the objects are monitored by at least one of the mobile sensors. However, when the mobile objects start moving they might not be monitored. For this reason, the mobile objects also need to move and we design this movement algorithm in this section of the thesis. Note that, we consider the speed of the mobile objects to be higher than the mobile sensors. Otherwise, the problem becomes trivial.

We compare our algorithm in terms of the percentage of the objects monitored with the increase of time and find that after a certain period of time this number becomes almost static and we can monitor a good portion of the objects continuously. As there are no other known local algorithms for this problem, we consider the algorithms designed for the dispersion of mobile sensor networks (Chapter 4) to compare and find that the latter does not perform well in this particular problem compared to our new algorithm.

1.2.6 Depth Adjustment of Underwater Sensor Networks

We propose a cellular automaton based algorithms for the depth adjustment of sensors in underwater sensor networks. Initially, mobile sensors are deployed at the surface of water. There are some relay stations deployed on the surface too. The mobile sensors

try to disperse in the water to improve the coverage of the underwater network. However, they might lose the connectivity with the surface stations while they move. We design some rules that help the sensors to move and maintain connectivity too. To the best of our knowledge, there is no such local algorithm for this particular problem. We consider different random initial configurations of the sensors and compare the final results of the sensors with a benchmark. We find that our algorithm performs very close to the benchmark [28].

1.3 Organization of the Thesis

The thesis is organized as follows:

- *Chapter 2:* We discuss some background information on wireless sensor networks, mobile wireless sensor networks, underwater sensor networks and cellular automata. We also define a statistical term that we use for our simulations later.
- *Chapter 3:* Our algorithm for the sleep-wake scheduling problem is presented in chapter 3. We present our algorithm with simulation results. We also discuss some related work and future research directions.
- *Chapter 4:* Algorithms for the autonomous deployments of the mobile sensors are presented in chapter 4. We present different variants of the algorithm and the performance analysis of the algorithms.
- *Chapter 5:* Cellular automaton based local algorithms for the synchronous gathering problem are discussed in chapter 5. Related simulation results and a

comparative study is also presented.

- *Chapter 6*: This chapter discusses the algorithms for minimizing chain length problem in details.
- *Chapter 7*: Our cellular automaton based algorithms for the monitoring mobile objects problem with simulation results are presented in chapter 7.
- *Chapter 8*: We study the depth adjustment problem of underwater sensor networks in chapter 8 and present the local algorithm with detailed simulation results.
- *Chapter 9*: Finally, we conclude the thesis with some future research directions in chapter 9.

Chapter 2

Preliminaries

In this chapter we present some general background on WSN, MWSN, UWSN, and recall some basics of statistical analysis that will be needed for our simulation results in the thesis. First we discuss wireless sensor networks in brief in section 2.1. We also discuss some of the research challenges of wireless sensor network in this section. The details of mobile wireless sensor networks with some research issues are discussed in section 2.2. Some background information on underwater wireless sensor networks is offered in section 2.3. We also present some of the basic definitions of cellular automata and discuss their usefulness for applications in section 2.4. Finally, we define the notion of confidence interval in section 2.5.

2.1 Wireless Sensor Networks

A wireless sensor network is a wireless sensing and networking technology that has attracted much attention because of its wide applications. A typical WSN consists of a number of tiny nodes called sensors. Sensors have some unique characteristics

compared to other networking devices. A simple sensor node has four basic components: a *sensing unit*, a *processing unit*, a *transceiver unit* and a *power unit*. It may have some additional units based on the applications, for example, a *location finding system*, a *power generator*, and a *mobilizer* [5]. The sensing unit is further divided into two parts: *sensors* and *analog-to-digital converters* (ADC). The processing unit, equipped with a small storage unit, helps the sensor node to collaborate with the other nodes in order to perform the assigned sensing jobs. The transceiver unit connects the sensor to the network. Some routing and sensing tasks need finding the locations of other sensors so some of the sensors might have a location finding system. Figure 2.1 [5] represents the components of a sensor. The arrow symbols represent the relationships between the components.

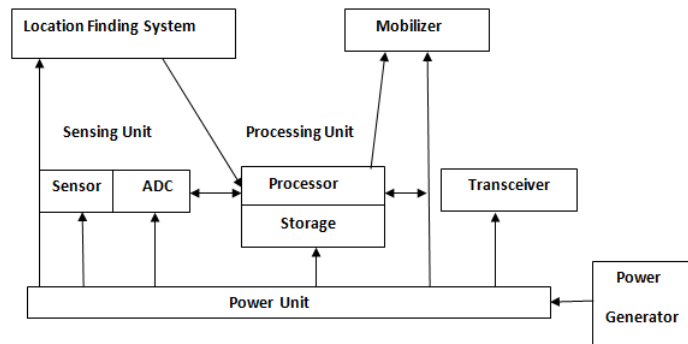


Figure 2.1: The Components of a Sensor

Sensor networks are designed to perform some high level information processing tasks. Some of the applications of wireless sensor networks are listed below [5, 84].

- Environmental monitoring.
- Industrial process control.

- Infrastructure protection.
- Military battlefield awareness.
- Security and surveillance.
- Context-aware computing.
- Health care system.

Based on the applications, the sensors are deployed randomly or deterministically to gather the information from the environment in a WSN. In a military battlefield, sensors are usually deployed randomly to gather the information while in an environmental monitoring system sensors are deployed deterministically in the critical places to monitor the aspects of the environment. Normally a large amount of sensors are deployed in the environment to sense the data.

Unlike other traditional networking devices, sensors have limited power supply on their batteries. Moreover they have limited processing power, limited memory and limited transmission range. A WSN also has one or more base stations that send some queries and gather the data from the sensor nodes. A base station is also referred to as a data collector or a sink node [118]. Normally the base stations and the sensor nodes are fixed on their position but depending on the applications they can be mobile too.

The sensors in a WSN send the data to a base station after doing some local processing. This communication can occur directly if the base station is within the transmission range of the sensors. If the base station is not within the transmission range of the sensors then the sensors use *multi-hop communications* (using other intermediate sensors). A neighbor of a sensor is called *next hop* if the sensor uses that

neighbor to route a message. The intermediate devices are used to plan the routing of the messages to their destination. Since the sensors have limited storage capability, they can not store the whole topology of the network. Figure 2.2 shows a multi-hop routing in a WSN. The circles denote sensors.

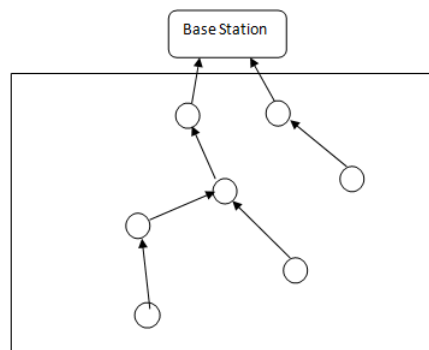


Figure 2.2: Multihop Routing

Since the energy of the sensors is limited and they use multi-hop communications to reach the base station, energy efficient routing protocols and the placement of the base station play a vital role in a WSN. Designing efficient algorithms for these problems involves other fields of research, such as computational geometry, approximation algorithms, integer linear programming, etc.

Next we present an overview of the routing schemes for WSNs.

2.1.1 Energy Constraints and Routing in WSN

Due to limited energy supply, preserving energy is usually the most important factor in determining the lifetime of a sensor network. The energy of a sensor decays when a sensor gathers data and sends data. Even in a asleep mode the sensor needs a power supply. We consider a sensor node to be dead when there is no energy left in

it. Normally a network lifetime refers to the time until the first sensor of the network dies. Sometime it denotes the time until a portion of the sensors die. Network lifetime is an important performance metric of a WSN and it depends on the energy of the sensors. Normally we can assume that the base station has a constant energy supply. Because of multi-hop communication, a sensor not only sends its own data but also it acts as a relay node to send data originating from other nodes to the base station. For this reason, sensors that are quite near to the base station die early [8]. The remaining energy of a node at any given time is called the residual energy.

A typical routing protocol, besides finding routes between given nodes, needs to take care of various tasks such as determining network topology and detecting changes in it, and maintaining network connectivity. As a sensor dies, the topology of the network can be changed and it might take more energy for the other sensors to recalculate the route and send the messages to that route. Therefore designing an energy efficient routing scheme is a very challenging task in a WSN. In any energy efficient routing protocol one of the main concerns is to use the minimum energy to send and receive data. As in most cases the base station is the destination of the message so the placement of the base station also plays a vital role in the design of an energy efficient routing protocol.

Selecting the next hop for sending data is the basic task of any routing protocol. Any node of the network selects its next hop based on its stored information. We can classify the routing protocols of wireless sensor networks into five main categories based on the next hop selection process [1]. The categories are: content-based routing protocols, probabilistic routing protocols, location-based routing protocols, hierarchical-based routing protocols, and broadcast based routing protocols.

Now we explore some of the most important routing protocols of each category.

Content-based Routing

In **content-based** or **data centric** routing, the next hop is determined based on the query content. In content-based routing, the base station does not send a query to a specific node. It normally broadcasts the queries and waits for the data from the sensors in the selected region. In a WSN it is not feasible to assign global identifiers to each of the sensors because of the density of the sensors in the network. In a commonly used approach every sensor sends all the data within its transmission range. This typically causes problems with energy consumption, redundancy, data aggregation, etc. Details of some of the content-based routing algorithms can be found in [1, 2]

Probabilistic Routing Protocol

Energy aware routing [97] is a probabilistic routing protocol designed to add load balancing and robustness to the routing process. In this protocol the next hop is selected based on some probability analysis. This protocol extends the life time of a network by load balancing.

Location based routing

Location based routing is used when a node knows the position of its neighbors and the destination. It simply sends the message to its neighbor that is close to the region of the destination. **GEAR (Geographical and Energy Aware Routing)**, **Greedy Perimeter Stateless Routing (GPSR)**, and **Geographical Adaptive**

Fidelity (GAF) are other location based routing protocols [1, 2].

Hierarchical-based routing

In a **hierarchical-based routing** protocol, a multi layer architecture is used. Instead of receiving the data from all the sensors the base station receives the data from some of the representatives of those sensors. These representatives are called gateways or cluster heads. Sometimes they are considered less energy constrained and called relay nodes. **Low Energy Adaptive Clustering Hierarchy (LEACH)** [56] is the first protocol of this category. Some other protocols of this category are **Power Efficient Gathering in Sensor Information System (PEGASIS)**, **Hierarchical-PEGASIS**, **TEEN**, and **APTEEN** [1, 2].

Broadcast-based routing protocol

The **Minimal Cost Forwarding Algorithm** [117] is the only routing protocol of this category. The basic idea of this protocol is that, upon receiving the data, each node decides whether it should forward the data or not. Every node on the path checks whether it is in the minimum route or not. If it is in that route then it just broadcasts the message, otherwise it drops the message. In this protocol every sensor only stores its own cost to reach the base station.

2.1.2 Performance Evaluation

Along with some typical performance metrics of traditional networks (such as, routing cost, bandwidth, response time, etc.) there are several additional metrics used to measure the effectiveness of a wireless sensor network. Some of the most commonly

used metrics are listed below [39].

- *Coverage*: The total area covered by the network at a given period of time. The coverage of the network starts decreasing once the sensors start dying.
- *Network Life Time*: The survival time of the network is called the network life time. Different definitions of the network life time are used in the literature. In some cases, it is considered as the time until the network becomes disconnected. On the other hand, other definitions such as “until a fraction of sensors die” or “until all the sensors die” also have been used. The definition of the network life time is varies with applications.
- *Connectivity*: As connectivity is an important aspect for routing, it is considered as an important metric. Once the network is disconnected, some parts of the network can not communicate and send the data.
- *Residual Energy*: As the time goes, the energy of a sensor decays and the sensors die when all of their energy is finished. Residual energy of the network at any given time is an important metric to measure the performance of the network.
- *Alive Sensors*: As soon as a sensor dies, we can lose some coverage, so the number of alive sensors are very important in the performance of a sensor network.

2.1.3 Research Challenges

As the limited energy is a very important aspect of the sensors, maximizing the network lifetime while maintaining a competitive coverage is a well studied research problem in wireless sensor networks [109, 80]. There are different algorithmic solutions for this problem. Most of these solutions are either centralized or distributed.

Centralized solutions have some problems in scalability and are not applicable for different applications of the sensor networks, for example, animal monitoring, battlefield, etc. On the other hand, in a distributed solution, a sensor needs to pass a lot of messages among its neighbors that incur energy too. However, in a local solution the sensor needs very little amount of information to compute and is an ideal candidate for the different optimization problems of sensor networks. One such solution is presented in [39] for the coverage problem of the wireless sensor networks. We propose a better algorithm for the same problem in chapter 3.

2.2 Mobile Wireless Sensor Networks

2.2.1 Motivation

Some of the advantages of using mobile sensor networks are [82, 11, 76]:

- In different applications, for example, military battlefield, object monitoring, it is not possible to deploy the sensors optimally. Adding mobility in the network gives us the flexibility to deploy the sensors randomly at the beginning and the sensors then move themselves to the optimal positions.
- Mobility also can improve the coverage.
- In a static sensor network, due to node failure, the network can be disconnected. Mobile sensors can be used to re-established the connectivity of the network.
- If the sinks of the network are stationary, then due to the traffic load, the sensors close to the sinks can die earlier. Introducing mobility to the sinks can solve this problem.

Though sensor networks have been used successfully in different applications of our daily life, there are some critical issues that can not be solved using static sensor networks, for example, deploying the sensors deterministically in environmental monitoring or in a battlefield. To overcome these issues, mobility is included in the sensor networks.

2.2.2 Architecture

In a typical mobile wireless sensor network any component of the network, for example, sensors or base station(s) can have moving capability. They can be mounted on different types of unmanned vehicle. Mobile wireless sensor networks can be classified into three categories [11].

- *Flat or Planar*: A flat network consists of a set of heterogeneous devices that can communicate over the same network in an ad hoc manner. Some of these devices can be stationary too.
- *Two-tier* A typical two-tier network consists of static and mobile nodes. Mobile nodes make an overlay network that helps to move data within the network. The nodes of this overlay network can have more processing power, longer communication range and higher bandwidth. These nodes can be used to maintain the connectivity of the network.
- *Three-tier* This type of network is used in different applications simultaneously. In a three-tier network three types of nodes are available, static nodes, mobile nodes and access points. The mobile nodes gather data from the static nodes and pass the data to the access points.

2.2.3 Differences Between WSNs and MWSNs

Introducing mobility to the nodes of the network can change different aspects of the networks. The aspects that can be changed due to adding mobility are [82, 11]:

- *Localization*: As in the static sensor networks, sensors do not move once they deploy, it is easy to obtain the location information of the sensors. On the other hand, in case of a MWSN, as the sensors move within the network additional time, energy, and localization techniques are necessary to obtain the locations of the sensors.
- *Network topology* As the sensors in a MWSN can move, the routing tables are updated with time. Most of the typical routing algorithms of WSN cannot adopt the dynamic change of the routing tables. Different routing techniques of mobile ad hoc networks (MANET) can be adopted in these cases.
- *Power Consumption* Power consumption in a MWSN is much higher than a WSN because mobility needs much more energy compared to the communication energy. Normally, a much larger energy reservoir or self-charging capabilities are used in a MWSN.
- *Network Sink* In a typical WSN, the sinks are static. However, in a MWSN sinks can be mobile and they can visit the sensors to gather the data. So data gathering techniques can be quite different in a MWSN.
- *Additional Complexity* In addition to the hardware complexity, there are significant algorithmic complexity issues involved in deploying mobile sensors. The design of algorithms for the optimal movement plan for the nodes in a MWSN are quite challenging.

2.2.4 Research Challenges

Mobility is useful in different applications. However, it also introduces new research questions. Most of them are related to the movement of the sensors because in most of the applications of the MWSN, sensors are deployed randomly in the network. However, they need to move within the network for various purposes, for example, increasing coverage [72], repairing connectivity [48, 102, 35], or gathering in different geometric shapes [12, 46, 65]. We consider these issues in Chapters 4, 5, 6, and 7.

2.3 Underwater Sensor Networks

Underwater networks are a new and unexplored area of research though underwater communication has been experimented with since World War *II*. Advances of sensor technology give us a chance of using underwater sensor networks in different applications. Some of the applications of underwater sensor networks (UWSN) are [4]:

- Environmental monitoring: pollution monitoring, monitoring ocean current and winds, improved weather forecast, detecting climate change, tracking of fishes and other important objects in water, etc.
- Undersea exploration.
- Disaster prevention.
- Navigation.
- Distributed tactical surveillance and mine reconnaissance.

Design Challenges

Radio waves need large antennae and high power to propagate under water. So instead of radio waves, acoustic signals are used in a typical underwater sensor network which is quite slow compared to the radio signals. Moreover, it needs high precision laser beams. Some of the design challenges of underwater sensor networks are [55, 4]:

- Centralized or deterministic deployments of the sensors are quite hard.
- Bandwidth is quite limited.
- Propagation delay is high and variable.
- High bit error rates.
- Battery of the sensors can not be recharged and solar energy can not be used.
- Sensors are prone to failures because of fouling and corrosion.

Architectures of UWSN

There are three types of architectures considered in the literature [4].

- **Static Two-Dimensional Networks:** In this type of network, stationary sensors are deployed at the bottom of the lake or sea. The nodes of the network are interconnected with underwater sinks with the help of acoustic communication. Each sink is equipped with two transceivers (horizontal and vertical). The horizontal transceiver is used to communicate with sensor nodes and the vertical one is used to communicate with the surface stations.

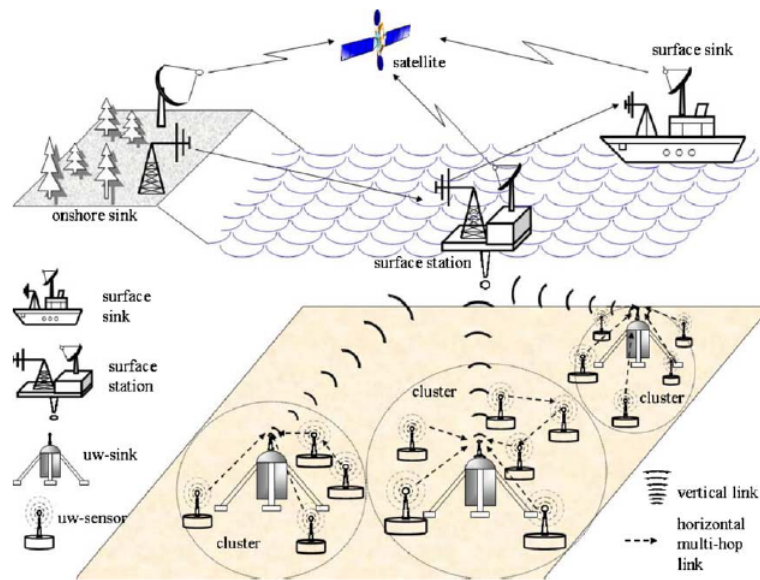


Figure 2.3: Architecture of a Two Dimensional Underwater Sensor Networks [4]

- Static Three-Dimensional Networks: The nodes are deployed at the different depths of the network.
- Three Dimensional Networks with Autonomous Underwater Vehicles: This type of networks consist of static and mobile sensors at different depths.

Two and three dimensional underwater sensor networks are shown in Figures 2.3 and 2.4 [4].

Research Challenges

One of the important optimization problems in UWSN is to decide on the movements of the mobile nodes. After deploying the sensors at the surface level the sensors might need to adjust their depth in the water autonomously to improve the coverage of the network while maintaining the connectivity with the surface station [3, 44, 45].

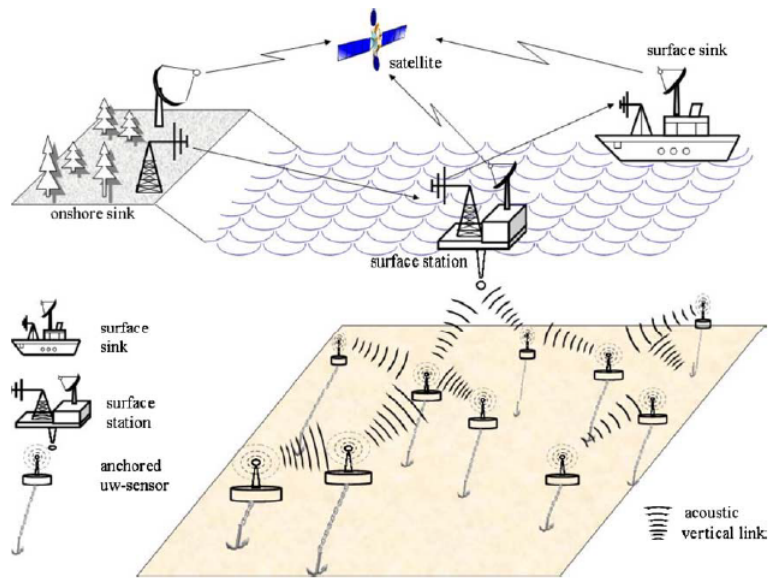


Figure 2.4: Architecture of a Three Dimensional Underwater Sensor Networks [4]

Though there are few local algorithms for two dimensional mobile sensor networks, there is no local algorithm available for the similar problem in a UWSN. We consider a local algorithm for this problem in chapter 8 of this thesis.

2.4 Cellular Automata

Cellular Automata (CA) are discrete dynamic systems in which the space, time, and states are discrete [61, 25]. They were first proposed by Von Neumann as a formal model of self producing biological systems and they became popular when Conway [50] implemented the famous “Game of life” using CA.

A cellular automaton works on a grid of cells. The grid can be more than one dimensional. Each cell of the grid can be in one of n possible states and each cell has a set of neighboring cells. Two commonly used neighborhoods include *Moore* and

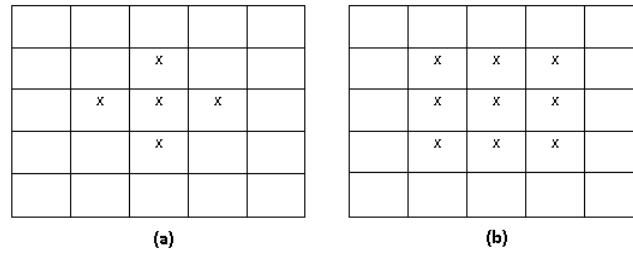


Figure 2.5: (a) Von Neumann Neighborhood, (b) Moore Neighborhood

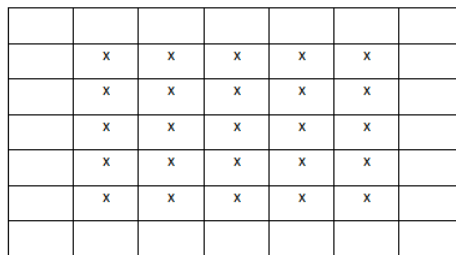


Figure 2.6: Moore Neighborhood with Radius-2

Von Neumann neighborhood. Figure 2.5 shows these neighborhoods (marked as “x”) for the central cell of the grid having radius 1. The Moore neighborhood of a cell consists of nine cells including itself while the Von Neumann neighborhood does not include diagonal cells. The grid can be finite or infinite. Note that, the boundary cells can have a smaller number of neighbors than the other cells.

Figure 2.6 shows the Moore neighborhood of the central cell for radius-2.

We can define a 2-dimensional cellular automaton as a quadruple, $A = (C, S, \delta, N)$ where C is the grid of cells, S is the set of possible states of the cells, δ is the transition rule of the automaton and N is the neighborhood of a cell. At time t , each cell of C is assigned a state of S . The state of a cell $c \in C$, at time $t + 1$, is determined via the transition function δ depending on the current state of c and the states of cells in the neighborhood of c at time t .

The game of life [50] works on a two dimensional grid and uses the Moore neighborhood. Each cell of the grid can be in either “dead” or “alive” state. The transition rules are as follows:

- If a cell is in the alive state and exactly 2 or 3 of its neighbors are alive then it continues to be alive; otherwise, it dies.
- If a cell is in the dead state and exactly 3 of its neighbors are alive then it changes its state to alive, otherwise it remains dead.

In general, a cellular automaton starts at time, $t = 0$ when each cell can be in one of the finite number of states. This is called the initial configuration. The transition function takes the states of the neighborhood of the current cell at time t as inputs and on the basis of them gives the state of the current cell at time $t + 1$.

2.4.1 Formal Definition

Here we give a formal definition of CA which follows [85, 19].

A cellular space is an infinite k -dimensional space of automata cells together with a neighborhood relation defined on this space and a finite set of possible states for each cell S . The set of states S must include a distinguished “blank” state. The neighborhood of each cell is a finite set of nearby cells. The time basis of the system is synchronous ($t = 0, 1, 2, \dots$). Each automaton in the cellular space is specified by a transition rule T which gives the state of a cell at time $t + 1$ as a function T of its own state and the states of cells belonging to its neighborhood at time t .

Also, it is required of T that if a cell and all of its neighbors are in the “blank” state at time t , then the cell will be in the “blank” state at time $t + 1$. This property

guarantees that when the computation is started with all but a finite number of cells in the “blank” state, this property is preserved during the computation.

Finally, a cellular automaton is made up of a cellular automaton system together with an assigned initial state for each cell in the cellular space at time $t = 0$. The initial state of the system may only contain a finite number of cells which are in states other than the “blank” state.

2.4.2 Classification

Wolfram [113] proposes following four classes according to the behavior of CA rules.

1. Evolution on a random initial configuration leads to a homogeneous state.
2. Evolution on a random initial configuration leads to a set of separated simple stable or periodic structures.
3. Evolution leads to a chaotic pattern.
4. Evolution leads to complex localized structures which are sometimes long-lived.

Variants of CA

Cellular automata can be thought as different types by varying their features [93] : the geometry of the underlying medium which contain the cells, local transition functions, and the neighborhood of the cells. Some variants of CA are mentioned in the following:

- Geometry: Different dimensional CA are available, for example, 2-D, 3-D.
- Neighborhood: Von Neumann and Moore.

- **Local Rules:** The rules of CA are usually deterministic, however different probabilistic rules are also available. If all the cells use the same rule then the CA are called “uniform” otherwise “non-uniform”. If all the cells do not update at the same time then they are called an “Asynchronous CA”. A cellular automaton is called “Reversible” if the configuration, for all t , at time $t + 1$ has only one possible past configuration at time t . If the rules depend only on a total or sum of the neighboring states then it is called “Totalistic”. In totalistic CA, the states are integers. The game of life mentioned earlier is an example of a uniform, synchronous and totalistic CA.

2.4.3 Why CA in Sensor Networks

Four partially overlapping motivations for studying CA are mentioned in [61].

- A large number of simple locally connected cells serve as a powerful parallel machine.
- We can perform extensive and systematic simulations of physical systems.
- CA have been used as a theoretical tool for studying complexity and pattern formation.
- Cellular automaton is a new discrete dynamic approach to model different complex systems.

One of the unique features of the sensor network is locality. Each sensor has a small processor that is aware only of its neighborhood which is similar to CA.

2.4.4 Some Applications

CA Games

CA have been used to model and simulate different games. The most famous one is the “game of life” proposed by Conway *et. al* [21]. It illustrates how simple rules can be used to characterize the behavior of a highly complex system. CA have been used to model other games, for example, firing squad [81], firing mob [38], queen bee [90], iterated prisoners dilemma [95], etc.

VLSI Design and Testing

Typically 1-D CA have been used successfully in VLSI design and testing [41]. Local communication between the homogeneous cells can be taken as a strong argument to use CA in VLSI. In the VLSI context, CA have been used to generate pseudo random sequences, error correcting codes, private key cryptosystem, design of associative memory, aliasing and testing the finite machine.

Traffic Modelling

CA have been used to better understand car traffic flow through simulations. Nagel *et. al.* [83] propose a first cellular automaton model to simulate single lane highway traffic. In this model each car uses the same transition rules and moves v cells at each time interval. One probabilistic cellular automaton model is proposed in [20] which is called BJH model. As multiple highway lanes are more realistic, different CA models for multiple lanes have been proposed in [91]. More realistic versions have been proposed in [34, 53].

Wireless Sensor Networks

Cellular automata have been used in modeling wireless sensor networks. Cunha *et al.* [39] propose a cellular automaton based algorithm for sleep-wake scheduling of wireless sensor networks. They also have been used to design systems to protect the security of sensor networks[105, 64]. Other CA based algorithms for sensor networks are considered in [18, 106, 71].

Some other applications of CA are: parallel computing [15, 47, 79, 115, 108], physical and biological systems [104, 94, 26], environmental and ecological systems [98, 116, 119], image processing [89, 114], etc.

2.5 Confidence Interval

In this section we define the notion of confidence interval that we use later to discuss our simulation. For some basic terminology of the statistics and for a complete reference the reader is referred to [37].

A confidence interval gives an estimated range where the estimated range is calculated from a given set of sample data. It lies within a given range from a possible high to a possible low. A true mean is most likely to be somewhere within the specified range.

The confidence interval has two parts. One is an estimate of the quality of the sample for the estimate, known as the standard error of the mean, and the second is the degree of confidence provided by the interval specified, known as the standard Z -score. The standard error of the mean is the standard deviation divided by the square root of the sample size. The confidence interval of the mean is thus calculated by:

$$\text{Mean} \pm \text{appropriate } Z\text{-score} \times \text{standard error of the mean} \quad (2.1)$$

The Z -score is chosen from the level of confidence. For example, in the case of 95% confidence level we use $Z = 2.0$. On the other other hand, in the case of 99% confidence level $Z = 2.5$ is used.

Chapter 3

Sleep-Wake Scheduling of Sensors

3.1 Introduction

Typically, a large number of sensors are deployed in the environment to sense the data. Since the sensors are densely deployed, multiple sensors can sense or cover the same region. The sensor spends most of its energy to sense the environment and dies as soon as it loses all its energy. Different techniques have been applied to preserve the energy for as long as possible. One of these techniques is to let a part of the sensors sleep for a certain amount of time and let them awake only when necessary.

Much research has been done on such scheduling problems. One of the common techniques is to make a domatic partition of the sensors [109, 80]. In a domatic partition (set), some representative sensors are selected in a way that either a sensor is in that set or at least one of its neighbors is in the set. The idea is to let this set of sensors stay awake for a certain period of time so that the area of the network is covered and the life time of the network is also increased. As the domatic partition

problem is NP-hard [51], many approximation algorithms have been proposed [40, 10, 70, 27]. In such algorithms, it is assumed that the network is connected. Once the network is disconnected the algorithm is of no use. Different strategies to solve energy efficient coverage problems can be found in [23].

There are some algorithms to solve the sleep-wake scheduling problem of a WSN using cellular automata [39, 71, 18]. In every case, they consider a radius 1 neighborhood of the sensors. In this chapter, we consider a radius 2 neighborhood to solve the sleep-wake scheduling problem of a wireless sensor network. Moreover, we have used two different energy levels for sensing and communicating among neighbors. To make the comparison with the earlier radius 1 algorithms realistic, in our algorithm the sensing radius remains one and the larger radius two neighborhood is used only for communicating with neighbors. Naturally, the energy used for communication depends on the square of the communication radius, and this is taken into account when calculating the energy use of sensors. Furthermore, our algorithm solves the sudden falls of the coverage of [39]. We developed a CA simulator and compare the different rules to implement the CA model. Recently a variant of our algorithm is considered in [69]

Object detecting in a WSN is one of the well studied research problems [75, 13, 67, 14]. In an animal behavior monitoring environment or in a military field, sensors are usually deployed to detect animals or enemies, respectively. To this end, algorithms are proposed that increase the probability of success. In such a situation network life time is also an important factor. As soon as the network dies, we cannot detect the object any more. Later in this section, we also show that our CA model increases the network lifetime and can detect more objects than the earlier algorithms based

on CA models.

This chapter is organized as follows: In section 3.2, we describe our cellular automata algorithm. We define the coverage and the object detection problems and discuss how we use our algorithm to solve these problems in section 3.3. In section 3.4, we describe our simulation tool and the results and finally summarize in section 3.5.

3.2 Representing a Wireless Sensor Network as a Cellular Automaton

In the following, we extend the CA algorithms considered in Cunha *et al.* [39]. We represent the sensors as the cells of a two dimensional grid. Each cell can have three states: awake, asleep and dead. A real value is associated to each cell representing the remainder of the available battery for that sensor. At each time point, the value is decreased correspondingly, depending on whether the sensor is awake or asleep. When the value reaches 0 (or a negative value), the sensor enters permanently into the dead state. Initially, after the deployment, we set the states of a subset of cells as awake. In the awake state, the sensor can sense its region (radius 1 neighborhood) and communicates with its neighbors. For communication with neighbors, a sensor has, depending on the algorithm used, either radius 1 ([39]) or radius 2 neighborhood (our algorithm). The energy consumption when using the radius 2 neighborhood is larger (square of the energy consumption for the radius 1 neighborhood). While using a large communication radius increases the energy consumption, having more information about the states of nearby sensors may enable the network to make better decisions on which set of cells should be kept awake.

In practice, the sensing range and the communication range of a sensor can be different in a WSN [23]. In the asleep state, a sensor cannot sense and communicate with any of its neighbors. However, as explained below, a sensor in the asleep state periodically wakes up and communicates with its neighborhood to determine whether or not the sensor should go to the awake state. A sensor that runs out of energy goes permanently to the dead state.

We use only rules that count the number of awake sensors. The CA transition rule is applied only periodically. For $i, j \geq 0$, we define i/j rules by setting:

- A cell in the awake state that has at least i neighbors awake, goes to the asleep state; otherwise it remains awake.
- A cell in the asleep state that has less than j neighbors awake, wakes up; otherwise it stays asleep.

Thus, at periodic intervals, when the transition rules in a particular cell are applied, the cell wakes up and checks its neighborhood. For this time period the cell consumes energy (the energy consumption rate depending on the neighborhood radius as described later in the simulation and results section).

Note that, when counting the number of awake cells in the neighborhood, it does not make a difference whether the remaining cells are asleep or dead. Since a sensor in the asleep state is not using energy for communication, it appears to its neighborhood as a dead sensor. Thus, in order to keep the algorithm realistic, the CA rules cannot distinguish between sensors that are asleep and those that are dead. Though the boundary cells can always have less than i or j sensors for some rules (for example, 3/3, 3/4, 4/4, etc.), for simplicity we use similar rules for all the cells.

The algorithm terminates when all sensors are in the dead state.

3.3 The Coverage and the Object Detection Problems

In this section, we describe two problems for which we apply our algorithm: coverage and object Detection.

In a WSN, the area covered by the awake sensors is called the coverage of the network. A sensor in the awake state is considered to cover the sensors that are within its sensing radius (radius 1 neighborhood in our paper). Sensors in the asleep or dead states do not provide any coverage. A network, where a large number of cells is awake, provides good coverage, but due to the increased energy consumption many cells start to die out which then makes it impossible to maintain the coverage level. The goal of a coverage algorithm is to provide coverage at an acceptable threshold (measured as a percentage of the area covered) with as few sensors awake as possible. The decisions on which cells are asleep/awake need to be made locally, which makes CA a good model for this problem. What is an acceptable threshold naturally depends on the particular applications we have in mind. In general, the solutions to the coverage problem can be evaluated by considering the resulting coverage percentage as a function over time.

In the object detection problem, the network is used to detect various objects (such as wild animals, or enemy soldiers) that enter the network area. While the type of the movement may depend very much on the particular applications, here we assume that the objects move randomly. In one set of experiments, the objects change direction

at each time step, and we consider another variant where the objects move in one direction a fixed amount of time before making a random choice of a new direction. As far as we know, CA algorithms have not been previously considered for the object detection problem, and naturally for more detailed results, further work can consider objects whose movement is not random.

Note that, if we require that the object be observed at all times, the problem setting would become more similar to the coverage problem. Furthermore, when discussing the object detection problem, we assume that the sensors have sensing radius zero, that is, an object is detected only when it comes into the same grid point with an awake sensor. Choosing the sensing radius is just a matter of scaling. If we use a larger sensing radius, almost all objects will be detected even with very conservative algorithms that keep only a small portion of cells awake.

We identify the best transition rules for the coverage problem and the object detection problem both with respect to the radius 1 and radius 2 neighborhood and compare these with each other.

3.4 Simulation and Results

We developed a simulator in Python Version 2.7 to compare the different rules of cellular automata of our algorithms. We consider a 2-D grid of size 150 (150×150) and consider a network of 22500 sensors.

In our algorithm, we use the same energy values as used in [39, 32]. We use 0.8 J energy as the initial energy of each sensor. For the radius 1 neighborhood CA, we use 0.0165 J as the awake value (the amount of energy spent by a sensor in the awake

state) and 0.00006 J as the asleep value (the amount of energy spent by a sensor in the asleep state). For the radius 2 neighborhood CA, we use higher values when the sensor communicates (0.027 J for the awake sensors and 0.00036 J for the asleep sensors). Note that, while in the asleep state, the sensors can be viewed to consume energy at the same rate independently of the neighborhood size. At the periods, when the sensor checks the transition rule, it has to count the awake sensors in the large neighborhood. For this reason we have increased the energy consumption in the asleep state.

The sensors can deplete their energy too early if they communicate with their neighbors in every time period. For this reason, at each time period, we randomly select some sensors which will communicate with their neighbors. We draw a number between 0 and 1 for every sensor and if the number is smaller or equal to 0.2, then that sensor communicates with its neighbors and checks whether or not it should change state following the appropriate i/j rule .

The algorithm works as follows:

- Initially, a small number of sensors are awake instead of all sensors. We pick a random number between 0 and 1 for each sensor. If the number is less than 0.1 then we set the status of the sensor as awake otherwise asleep. If we start with more sensors awake (more than 10%), then initially we can get more coverage but after some time, the sensors will verify their neighbors and will find that there are enough sensors awake. So, almost at the same time, all of them will go to the asleep state and there will be a sudden fall of the coverage as in [39]. Selecting a small number of sensors, does not solve this problem entirely. We use a probabilistic technique to solve the problem entirely that is discussed later

in this section.

- Instead of the timers used in [39, 32], each sensor is probabilistically selected (with 20% probability) to verify its neighbors. If we use 10% probability, then each sensor checks its neighbors less frequently. For this reason, in practice, network life time is increased but we get less coverage than the previous one. On the other hand, if we use 30% probability, then we get more coverage initially as the sensors verify neighbors more frequently but the network life time is shorter than the one that we get if we use 10% probability.
- We force sensors that have stayed awake for longer periods to probabilistically go to the asleep state, even if their neighborhood has too few awake sensors to put them to sleep according to the normal transition rules. We set a threshold value 0 to each sensor when it is in the asleep state. We call this as the probability of going to the asleep state. Whenever any sensor changes its state to the awake state this value starts to increase and each time period (as long as it remains in the awake state) it increases by 0.05. So at each time period, we randomly select any number between 0 and 1 for each awake sensor and if the number is smaller or equal to this threshold then we forcefully set the status of the sensor to the asleep state.

3.4.1 Results for the Coverage Problem

We consider radius 1 and 2 neighborhoods with rules (1/1, 1/2, 1/3, 2/2 and 3/3) for the coverage problem. We have run 100 experiments on each rule and the average results are shown in Figures 3.1 and 3.2. We compare the performance of the different algorithms over time by computing the area under the curves. Naturally this is not

| Area Radius 1 (1/1) <i>Unit</i> ² | Area Radius 2(1/1) <i>Unit</i> ² |
|--|---|
| 5790034 | 6088327 |

Table 3.1: Area Under the Curves of Figure 3.3

the only possible measure, and depending on particular applications one might want to take only the part of the curve where coverage stays above a certain threshold. According to our measure, the 1/1 rule has the best performance for both radius 1 algorithms and the radius 2 algorithms. The performance of these two algorithms is compared in Figure 3.3. From this comparison, we see that initially the rule 1/1 of radius 1 algorithm gives more coverage but after a certain period of time it loses the coverage more quickly than the rule 1/1 of radius 2 and the network life time of the radius 1 algorithm is shorter than the radius 2 algorithm.

We also try 4/4, 4/5, 5/5 and so on. Though these rules give us better initial coverage, the network lifetime in these cases is much less than the rules that are mentioned in Figures 3.1 and 3.2.

We calculate the area under the curves of Figure 3.3 and show it in Table 3.1. We find that the total area covered by the 1/1 rule of a radius 2 neighborhood is 5% more than the 1/1 rule of a radius 1 neighborhood.

We also analyze the confidence interval of the mean for these two curves. We use the 95% confidence level to calculate the values. So the confidence interval of the mean is $\text{mean} \pm 1.96 \times \frac{\text{Std}}{\sqrt{N}}$ [37], where *Std* is the standard deviation of the population and *N* is the size of the population ($N = 100$ in our simulations). Using this formula, we find that the confidence interval of the mean of our experiments is quite narrow

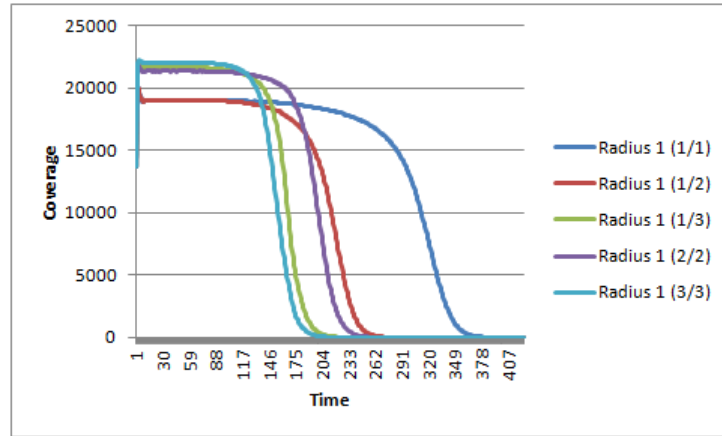


Figure 3.1: Comparison of the Radius 1 Rules of the Coverage Problem

(maximum 0.25%) compared to the average values. This is because the standard deviations of the values are very small. As the confidence interval is very narrow we do not show it in the figures.

We also consider two other metrics considered in [39]. One is the number of alive sensors (sensors are in awake and asleep states) in each time period and the other one is the amount of total energy left (residual energy) in the network at each time period. The total residual energy of the network is the sum of the residual energy of all the sensors. The comparisons are shown in Figures 3.4 and 3.5. In radius 2 algorithm, we need less sensors to be awake compared to the radius 1 algorithm at each time period. For this reason, in case of radius 2 algorithm, the sensors lose their energy slowly compared to the radius 1 algorithms.

In the above algorithms, we have implemented a probabilistic counter that forces sensors that stay awake for long periods to go to sleep before they completely run out of energy. The reason for the “forced” sleep periods is that without doing so, parts of

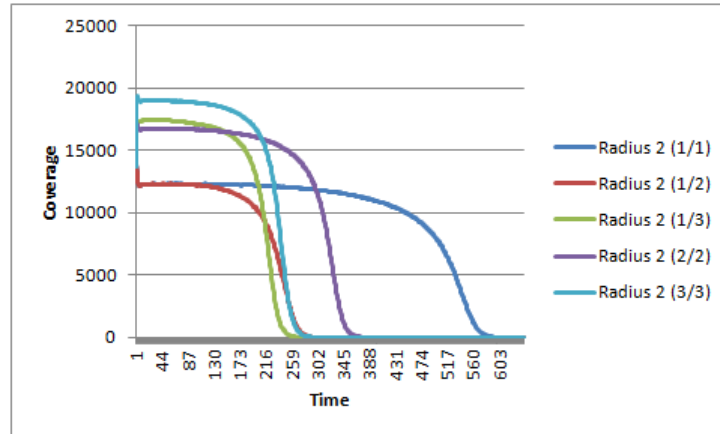


Figure 3.2: Comparison of the Radius 2 Rules of the Coverage Problem

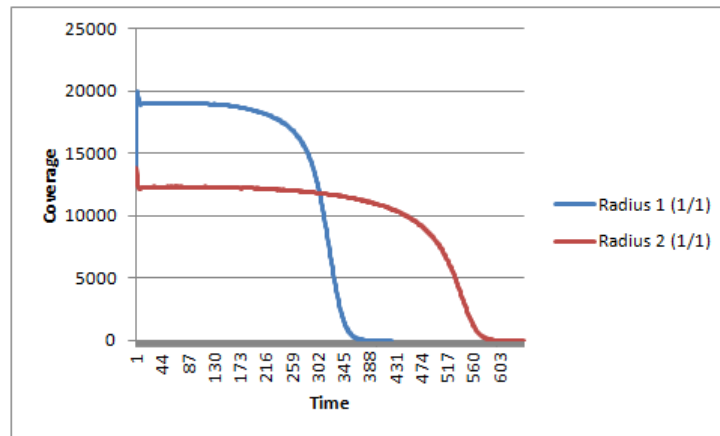


Figure 3.3: Comparison of the Rules of the Coverage Problem

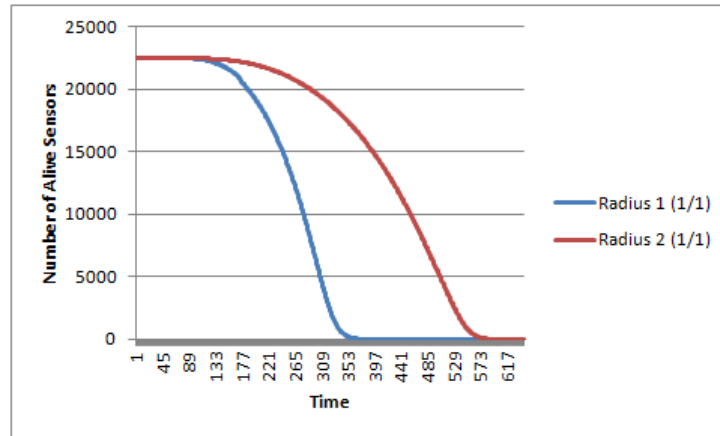


Figure 3.4: Number of Alive Sensors as a Function of Time for the Best Two Rules of the Algorithms

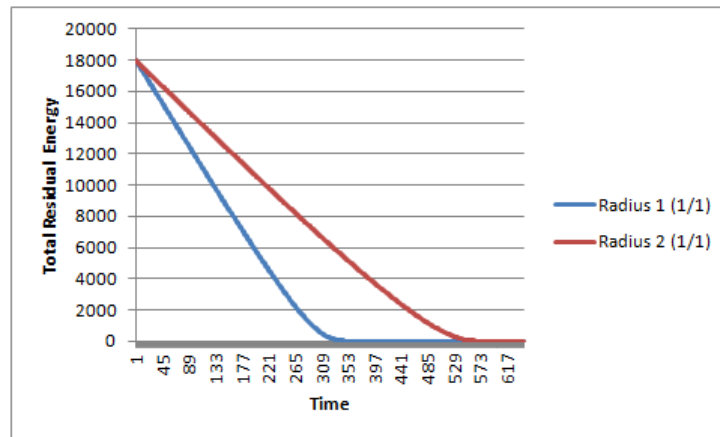


Figure 3.5: The Residual Energy of the Network as a Function of Time

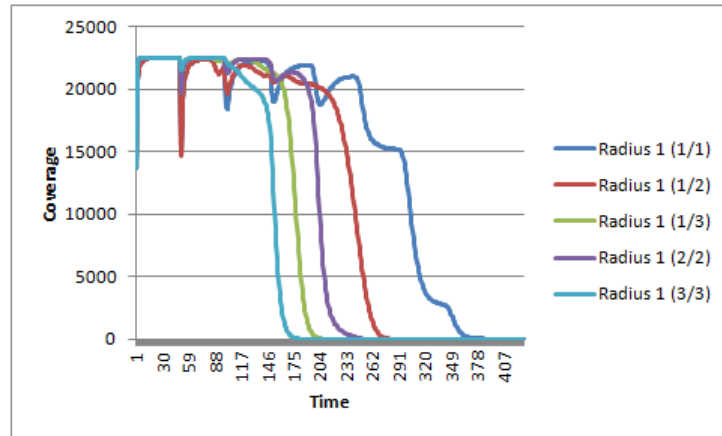


Figure 3.6: Comparison of the Rules of Radius 1 Neighborhood Without the Counter

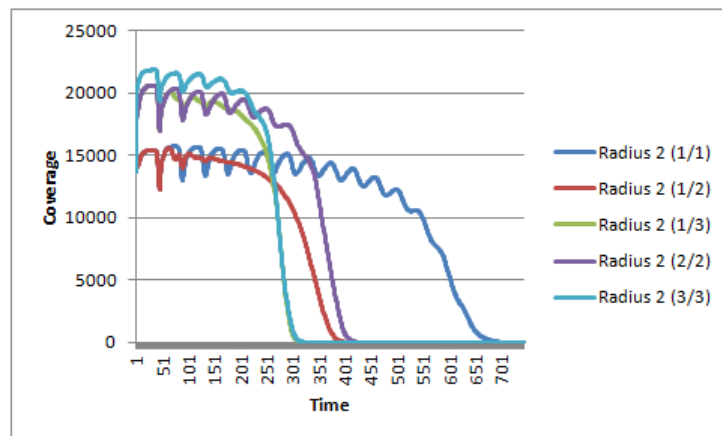


Figure 3.7: Comparison of the Rules of Radius 2 Neighborhood Without the Counter

the network seem to enter a steady state where particular sensors stay awake until they all die, more or less simultaneously, which causes a strange dip in the coverage. We also run 100 experiments on the rules of radius 1 and radius 2 neighborhoods without the probabilistic counter and the average results are shown in figures 3.6 and 3.7 respectively. We again find that rule 1/1 has the best performance for both radius 1 and radius 2 neighborhood and we compare the performance of these algorithms in Figure 3.8.

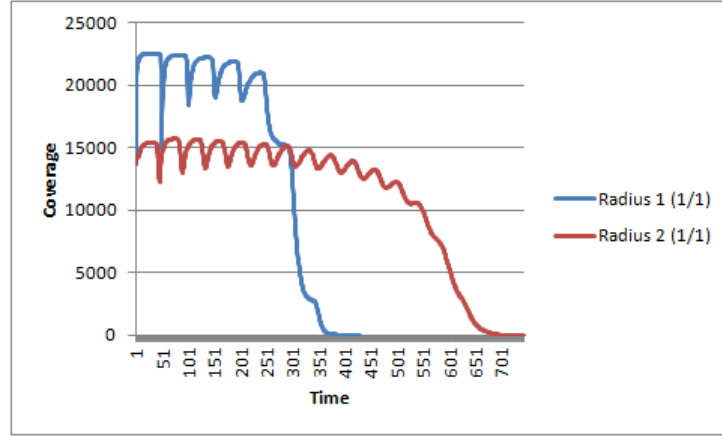


Figure 3.8: Comparison of the Rules Without the Counter

| Area Radius 1 (1/1) $Unit^2$ | Area Radius 2(1/1) $Unit^2$ |
|------------------------------------|-----------------------------------|
| 6371484.11 | 8227398.28 |

Table 3.2: Area Under the Curves of Figure 3.8

From Figure 3.8, we find that radius 1 algorithm gives us more coverage initially but radius 2 algorithm gives us more lifetime and a competitive coverage. We also calculate the area under the curves of Figure 3.8 and show it in Table 3.2. We find that the total area covered by the 1/1 rule of a radius 2 neighborhood is almost 30% more than the 1/1 rule of a radius 1 neighborhood.

3.4.2 Results for the Object Detection Problem

We also implement our algorithm to detect the objects in the WSN. We compare different rules of radius 1 and radius 2 neighborhood algorithms for the object detection problem and find that the 1/1 rule for both radius 1 and radius 2 neighborhoods

perform better than the other rules of the respective neighborhoods to detect objects. We perform two sets of experiments on the object detection problem. The movement patterns of the objects are different in two sets. Our goal of the experiments is to detect as many objects as we can.

Experiment 1

In the first set of experiments, at each time period, a constant number of objects are entered randomly in the network. The movements of the objects in the network are totally random, for example, just after entering the network, an object can go out of the network too. The simulation moves the objects randomly on an infinite two-dimensional grid. This means that the objects may exit the network permanently or may return at some later point in time. An object can move a maximum of one cell (in any direction, for example, left, right, top, down or diagonally) in one time step. We try to find out how many objects are seen by the sensors if we apply both algorithms. We randomly set the status of the sensors for both algorithms. A sensor is set to be initially awake with 10% probability.

We do our experiments on a grid size of 100, 125 and 150. So we have a total of 10000, 15625 and 22500 sensors respectively. In all the cases, we deploy either 5000 or 2500 objects in the network between 0 to 500 time intervals.

We find from the simulation that the best (1/1) radius 2 rule performs better than the best (1/1) radius 1 rule to detect the objects for all the cases. The radius 2 rules yield a much longer network lifetime than the corresponding radius 1 rules. So none of the objects, that come after the network life time of radius 1 can be detected by the radius 1 algorithm but still have a chance to be detected by the radius 2 algorithm.

We present the comparison results in Table 3.3. Each row of the table represents the average of 100 corresponding experiments. For example, in the case of a grid of size 150, 5 objects enter in the network in each time interval between 0 – 500 (total 2500 objects). We run 100 such experiments and find that among these 2500 objects, 442 and 786 objects are seen by radius 1 and radius 2 algorithms in average, respectively. Similarly, in all other cases, the best radius 2 (1/1) rule detects more objects than the best radius 1 (1/1) rule.

Figure 3.9 shows the comparison of another metric between the rules (1/1, 1/2 and 2/2) of the radius 1 algorithm for the object detection problem. Here we compare the amount of undetected objects at each time period among different rules. The figure shows the average results that we get from 100 experiments on the first row of Table 3.3. The X axis of the graph represents the time period and the Y axis represents the undetected objects (in percentage) at a time period. We calculate the percentage of undetected objects at time t as the ratio between the number of undetected objects at time t and the total number of objects that are in the network at time t . We find that initially all the rules cannot detect almost the same number of objects but overall rule 1/1 yields a better result than the others. Note that, we do not show the complete figure here. The results continue to show the same patterns after 257 time periods.

A similar comparison among the rules of the radius 2 algorithm is shown in Figure 3.10. Finally, a comparison between rule (1/1) of the radius 1 and rule (1, 1) of the radius 2 algorithm (the best rules) is shown in Figure 3.11 and we find that the radius 2 algorithm performs better than the radius 1 algorithm.

| Grid Size | Objects Per Interval | Total Objects | Objects Seen | Objects Seen | Objects Seen | Objects Seen | Objects Seen | Objects Seen |
|-----------|----------------------|---------------|----------------|----------------|----------------|----------------|----------------|----------------|
| | | | Radius 1 (1/1) | Radius 1 (1/2) | Radius 1 (2/2) | Radius 2 (1/1) | Radius 2 (1/2) | Radius 2 (2/2) |
| 150 | 5 | 2500 | 442 | 298 | 301 | 786 | 364 | 496 |
| 150 | 10 | 5000 | 884 | 594 | 601 | 1574 | 730 | 989 |
| 125 | 5 | 2500 | 438 | 295 | 299 | 773 | 361 | 491 |
| 125 | 10 | 5000 | 876 | 589 | 560 | 1548 | 723 | 981 |
| 100 | 5 | 2500 | 434 | 293 | 295 | 757 | 354 | 484 |
| 100 | 10 | 5000 | 868 | 581 | 577 | 1500 | 682 | 961 |

Table 3.3: Experimental Results on Object Detection Problem (Experiment 1)

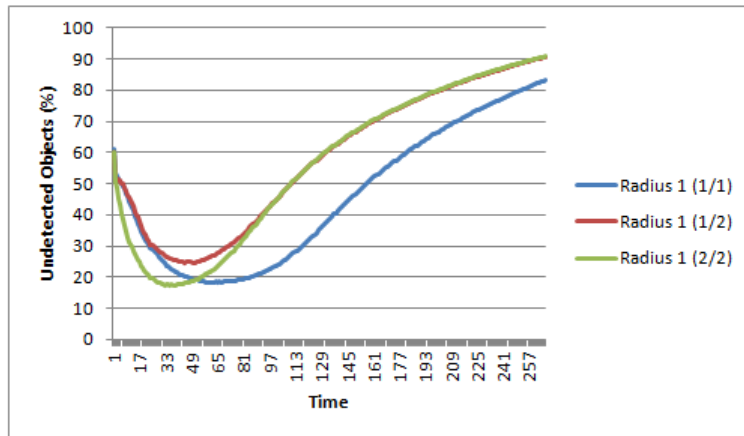


Figure 3.9: A Comparison of Radius 1 Rules on Object Detection (Experiment 1)

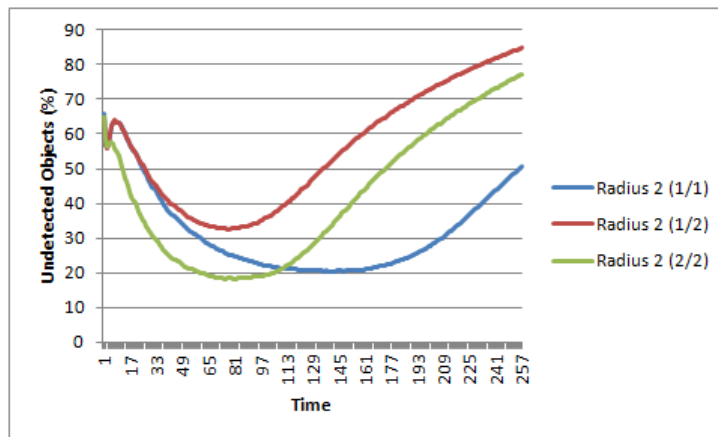


Figure 3.10: A Comparison of Radius 2 Rules on Object Detection (Experiment 1)

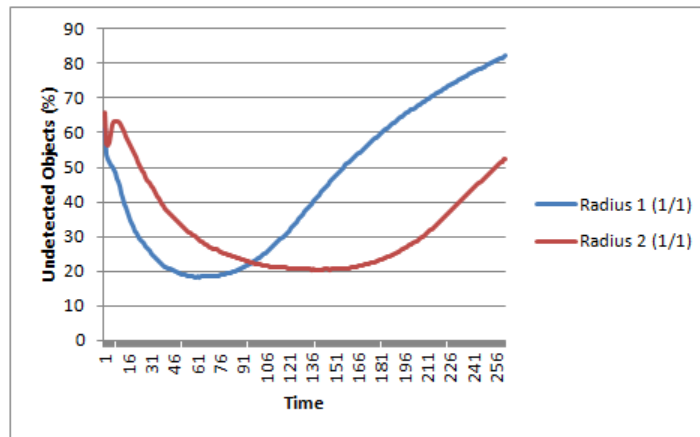


Figure 3.11: A Comparison of Radius 1(1/1) and Radius 2(1/1) Algorithms on Object Detection (Experiment 1)

Experiment 2

In the second set of experiments, we consider all the same simulation parameters as in experiment 1 except the movement patterns of the objects. In this case, initially the objects enter in the network in a random fashion but then they choose a random direction to move and move along that direction for a certain amount of time (we choose five time periods in our experiments). After that time, they again choose a random direction to move and follow that path. From these simulations, we again find that the rule (1/1) in both neighborhoods perform better than the corresponding other rules and the 1/1 rule of radius 2 neighborhood outperforms rule 1/1 of radius 1 neighborhood in all the cases. We also find that, though radius 1 algorithms detect almost similar number of objects that they detect in experiment 1 but the algorithms of radius 2 detect much more objects compared to the experiment 1. Table 3.4 displays the results of experiment 2. Figures 3.12, 3.13, 3.14 show the comparisons on similar parameters (2500 objects and time period 5) that are shown in figures 3.9, 3.10, and 3.11. From these figures, we also find that the 1/1 rule of radius 2 performs

| Grid Size | Objects Per Interval | Total Objects | Objects Seen Radius 1 (1/1) | Objects Seen Radius 1 (1/2) | Objects Seen Radius 1 (2/2) | Objects Seen Radius 2 (1/1) | Objects Seen Radius 2 (1/2) | Objects Seen Radius 2 (2/2) |
|-----------|----------------------|---------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| 150 | 5 | 2500 | 410 | 276 | 280 | 1690 | 995 | 922 |
| 150 | 10 | 5000 | 821 | 553 | 561 | 3241 | 1528 | 1766 |
| 125 | 5 | 2500 | 410 | 273 | 281 | 1671 | 780 | 905 |
| 125 | 10 | 5000 | 819 | 550 | 558 | 3269 | 1518 | 1779 |
| 100 | 5 | 2500 | 410 | 274 | 279 | 1722 | 795 | 924 |
| 100 | 10 | 5000 | 814 | 546 | 556 | 3161 | 1455 | 1698 |

Table 3.4: Experimental Results on Object Detection Problem (Experiment 2)

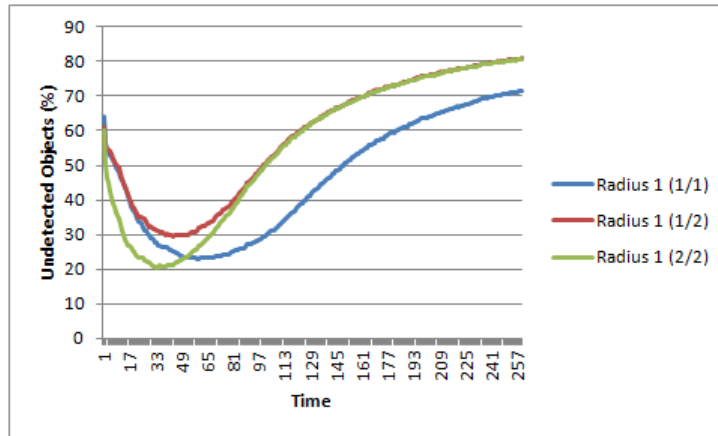


Figure 3.12: A Comparison of Radius 1 Rules on Object Detection (Experiment 2)

much better than the other rules.

3.5 Summary

We considered a radius 2 neighborhood cellular automaton model to solve the sleep-wake scheduling problem in a wireless sensor network. We showed that if we use the 1/1 rule of the radius 2 neighborhood algorithm then the network life time was

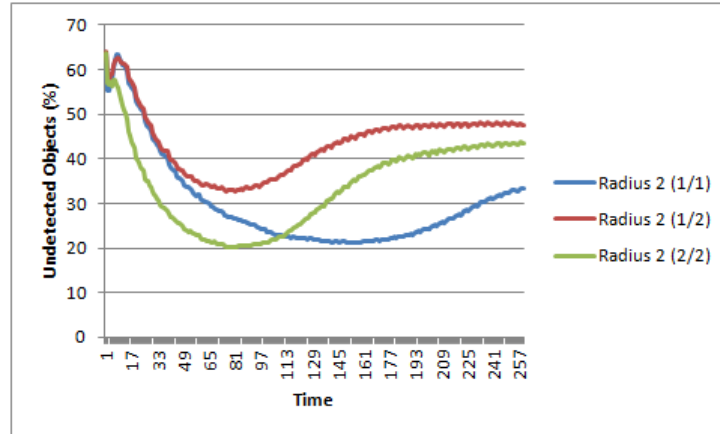


Figure 3.13: A Comparison of Radius 2 Rules on Object Detection (Experiment 2)

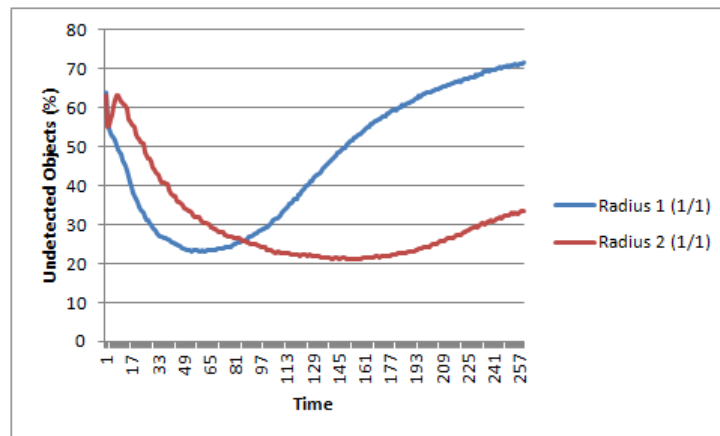


Figure 3.14: A Comparison of Radius 1(1/1) and Radius 2(1/1) Algorithms on Object Detection (Experiment 2)

increased and a good coverage of the network was also maintained. We also demonstrated by simulation that our algorithm was a good one for object detection problem.

Chapter 4

Autonomous Deployment of Mobile Sensors

4.1 Introduction

Mobile wireless sensor networks have a wide range of applications [72, 76] and there has been much research on this topic. The mobile sensors have the same general characteristics as in a sensor of a sensor network. Additionally, each mobile sensor has locomotion capability. One of the main reasons for using the mobile sensors is to improve the coverage of the networks. However, this leads to a very important research question. How one sensor can move so that it can maximize the coverage of the network?

Typically, a sensor is deployed in a network to cover some region of the network. The coverage of a network is defined as the area that is covered by its sensors. In different applications of MWSN, it is not possible to deploy the sensors deterministically so

that they can maximize the coverage. More commonly, they are deployed randomly and the sensors are required to disperse autonomously using algorithms to maximize the coverage of the network [76]. In different applications, along with the coverage, the preservation of the connectivity is also a major concern. Once the sensors try to move by themselves, they can break the connectivity. Connectivity is an important aspect of the network as it is used to route the data among the nodes. We call this problem *the Mobile Dispersion Problem*.

We first propose a new cellular automaton based algorithm that maximizes the coverage of a MWSN very quickly, involves fewer sensor movements when compared with an earlier cellular automaton algorithm [106], and is simpler than other existing techniques [36, 58, 111]. Though we have not considered the energy constraints on the movement of the sensors explicitly, the number of sensor movements gives a good approximate measure of energy use and gives a good measure of the usefulness of the algorithms in practice.

In the first set-up the sensors are restricted to an area. This means that, as long as the sensors are sufficiently dispersed (the algorithm works probabilistically), most of the network stays connected. We also consider a scenario where a number of sensors are initially densely deployed and the area is (at least potentially) unbounded. Thus, the algorithm needs to explicitly ensure that the connectivity of the network is preserved.

Connectivity preservation is an important aspect of different wireless sensor networks as it is necessary to route information within the network. In the second part of the chapter we also consider different CA based algorithms for connectivity preserving deployment of mobile sensors [31, 29]. In our experimental set-up we first consider a “regular” $n \times n$ square formation as an initial configuration of the network and

find an optimal solution using a deterministic algorithm. Finally, we study systematically more realistic random initial deployments of sensors and for the realistic initial deployments it is not equally easy to obtain an optimal solution. We need to add features to the algorithm that try to prevent the creation of “holes” in the network and also, in this context, it turns out to be useful to consider randomized variants of the algorithm. The problems caused by holes will be discussed in section 4.5.

There is another difference between the two variants of the problem. In the first variant, each cell can contain only one sensor at any time period. On the other hand, in the second variant a cell can contain more than one sensor.

We discuss some related algorithms in section 4.2. The system model of mobile wireless sensor networks is briefly discussed in section 4.3. Algorithms and related simulation results for the two variants of the problem are discussed in sections 4.4 and 4.5 respectively. Finally we conclude the chapter with some summary in section 4.6.

4.2 Related Work

Most of the solutions that have been proposed for this mobile dispersion problem are either global or distributed algorithms [68, 88, 58, 111, 36].

Vector based approach is a well known approach for deploying mobile sensors to maximize the coverage of the mobile sensor networks [59, 57, 86, 110, 121]. They all are inspired by the different physical models. In these cases, a sensor node determines its movement direction based on the force received from its neighbors. Howard *et.al.* [59] first propose a virtual force algorithm which is localized but it is not applicable for a discrete model and computing the directions is quite complex compared to

our cellular automaton algorithms. Zuo *et.al.* [121] propose a distributed virtual force algorithm for the dispersion of the mobile sensors. They use a combination of attractive and repulsive forces to determine the movement of the sensors.

Cortes *et.al.* [36] propose an algorithm for an analogous problem in robotics. In their algorithm, each robot (sensor in our algorithms) draws a Voronoi diagram and it moves to minimize its local uncovered areas by aligning its sensing range with the Voronoi region as much as possible. Some other Voronoi diagram based algorithms are proposed in [58, 111].

Barriere *et.al.* [17] propose a local algorithm for the uniform dispersion of autonomous mobile robots in a grid. However, their algorithm is quite complex and the algorithm does not work for the network having communication radius less than 4.

There are other local algorithms proposed for a variant of coverage problem [74, 73] called “focused coverage” where some regions of the network have more priority to be covered. The definition of this problem is quite different than the problem that we consider in this part of the thesis.

A simple approach has been proposed in [106] based on cellular automata to maximize the coverage of the mobile sensor networks. It does not explicitly consider the connectivity constraint and there are no techniques considered in the algorithm to preserve the connectivity. We discuss this algorithm in some more details when we describe our algorithm in section 4.4.

4.3 System Model of Mobile Sensor Networks

A set of n mobile sensors are deployed in a network. They are dispersed densely within the network. The sensors are homogeneous, *i.e.* they have same sensing (R_s) and communication (R_c) radii, where $R_c > R_s$. The sensing radius refers to the monitoring function of the network and a sensor is said to cover the cells within its sensing radius. On the other hand, the sensors can communicate with other sensors that are within their communication radius. The network is connected if any two sensors can be connected by a path of sensors m_1, \dots, m_k where m_{i+1} is within the communication radius of m_i , $i = 1, \dots, k-1$. The coverage of the network is the area covered by the largest connected component of the network. All the sensors move with the same speed. In our cellular automaton based algorithms, the sensors can move a maximum of one cell at a time. A random deployment of a mobile sensor network is shown in Figure 4.1. The small circles are the sensors.

4.4 Algorithms for Maximizing Coverage

We briefly discuss our algorithm for the first variant of the problem, *i.e.* maintaining connectivity of the network is not the main concern of this algorithm. The main goal of the algorithm is to increase the coverage. In this case, the sensors are deployed in a closed area and there are sufficiently many sensors so that when they are evenly distributed the majority of the sensors remain connected.

We consider a 2-D cellular automaton where the states of the cells indicate the presence or the absence of mobile sensors. The movement of the sensors is modeled by state changes. We consider different (R_c, R_s) pairs and the goal of the algorithms is

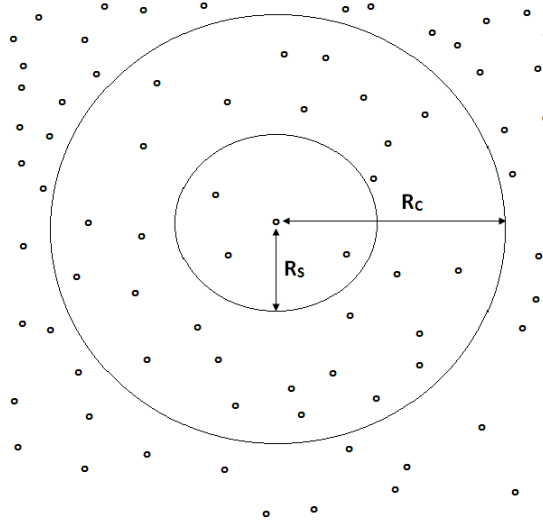


Figure 4.1: An Example of Mobile Wireless Sensor Network

to position the sensors in a way that maximizes the coverage.

The sensors can communicate and get information about the locations of nearby sensors up to R_c . During a given time period, a sensor spends much less energy to communicate with its neighbors than in monitoring its environment. So it is reasonable to assume that the communication radius is larger than the sensing radius even though the energy consumption increases with the radius.

In each time step, a sensor can move one cell in any direction. The algorithms use the information about the positions of the nearby sensors to determine the movement of the current sensor in the next time step. The goal of the algorithms is two-fold:

- to position the sensors in a way that maximizes the coverage,
- to minimize the movements of the sensors in order to conserve energy.

We propose the following cellular automaton model for this particular problem.

We consider an R_c neighborhood for each cell. We divide the neighborhood of a cell into the North East (NE), North West (NW), South East (SE) and South West (SW) quadrants. As indicated in Figure 4.2 for $R_c = 4$, each quadrant consists of 20 cells.

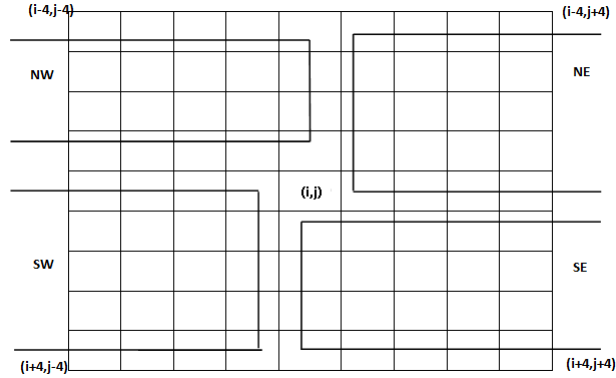


Figure 4.2: The Four Quadrants of a Cell (i, j) When R_c is 4

Now we describe the algorithm that is used to solve the problem.

1. Initially the sensors are placed in different ways (described in the simulation and results section) within the network. The state of the cell that contains a sensor is set to 1, otherwise 0.
2. Each time period is divided into two phases: odd and even (similar to [106]).
 - Odd Phase: The algorithms use two parameter values k' and k'' . In this phase, each sensor determines whether it should move (and where) or not.
 - * For each of the four quadrants $x \in \{ NE, NW, SW, SE \}$ of its current location, the sensor calculates a weight, t_x that is based on the number of sensors seen in that quadrant. The algorithm gives higher weight for sensors that are closer to the current sensor. For example, in case of $R_c = 3$, the neighbors that are distance 1 away are

assigned weight 3. On the other hand the neighbors that are distance 3 away are assigned weight 1.

- * Each sensor determines the maximum and the minimum value from t_x , where $x \in \{ NE, NW, SW, SE \}$.

- * If the minimum value is less than k' and the maximum value is more than k'' , then the sensor chooses a position randomly from the quadrant (one of the two positions of the quadrant that are adjacent to the current location of the sensor) that has the minimum value. In case of multiple minimum values, the sensor chooses a random quadrant. In the even phase, the sensor tries to move into that position.

- Even Phase: In this phase, a sensor moves to the cell that it has chosen in the odd phase if the following conditions are met:

- * The cell where it is moving should be empty.

- * No other sensor tries to move to that cell. Randomly one sensor is chosen if there are multiple candidate sensors for that cell. A radius $R_c = 2$ communication is enough to determine who are the candidates as the sensors move one cell at a given time.

- Once the sensor decides on its next location, the state of that location is set to 1 and the state of the sensor's previous location is set to 0.

By the (k'/k'') algorithm we mean the algorithm where the sensor movement condition requires that the minimum (respectively, maximum) weight computed for the quadrants is less than k' (respectively, greater than k''). We try different (k'/k'') rules, where k' and k'' are integers, $1 \leq k', k'' \leq 5$. We consider three different scenarios for

the grid of size 125 and compare the best rules of our model with an earlier algorithm [106] for which we use the name “COUNT”. The COUNT algorithm determines the movement of the sensors based only on a numerical quantity obtained by counting the numbers of nearby sensors at different distances and does not incorporate directional information into the numerical value.

Before explaining our simulation results, we briefly mention the differences between our algorithm and the algorithm from [106] (that we call the “COUNT” algorithm):

- Our algorithm divides the neighborhoods into four quadrants but “COUNT” does not.
- “COUNT” calculates only one k for each sensor, whereas our algorithm calculates four, one for each quadrant.
- Our algorithm forces the sensor to choose a future position from that quadrant that has minimum weighted value, while “COUNT” chooses a random one from all of its neighbors. So there is a possibility to choose a position where already enough sensors exist.

4.4.1 Simulations and Results

We implement all the algorithms using the programming language Python (Version 2.7) ¹. Next we describe the results of our algorithms for three different scenarios. We consider two metrics for the comparisons. One is the coverage and the other one is the number of moves.

¹I thank Sami Torbey for providing his source code for his cellular automaton model used in [106]

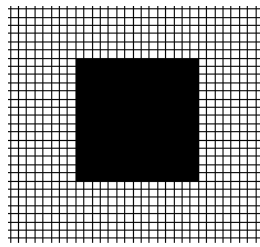


Figure 4.3: Scenario 1

We consider (R_c, R_s) pairs, where $2 \leq R_c \leq 4$ and $R_s = 1$ in the following simulations. The size of the grid is 100. We run all the experiments 10 times and report the average results.

Scenario 1

We put 225 mobile sensors as a cluster in the middle of the grid as in Figure 4.3. One sensor can cover at most 9 cells. So one can divide the grid into 3×3 blocks and put one sensor in the middle cell of each block and get the optimal, which is $9 \times 225 = 2025$.

We show the results for the best rules in Figures 4.4 and 4.5 in the case of $R_c = 4$ and $R_s = 1$. From the simulation, we find that rules (3/2) and (3/3) give us near optimal coverage. In terms of sensor movements, we find that 3/3 reaches the near optimal solution with fewer sensor movements.

We also find that, at time 100, the (3/3) rule reaches near optimal coverage. We show the positions of the sensors at that time in Figure 4.6. We also show the positions of the sensors of “COUNT” at time 100 in Figure 4.7 where total coverage is much less.

We can also put an upper bound on the sensor movements. For example, we put three different upper bounds (30, 50 and 70) on sensor movements for this scenario. So no

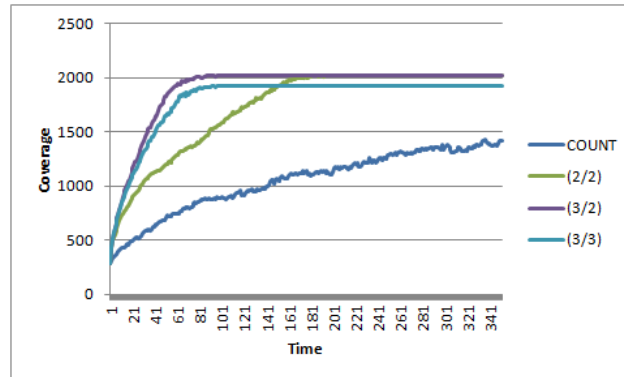


Figure 4.4: Scenario 1: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 4$ and $R_s = 1$

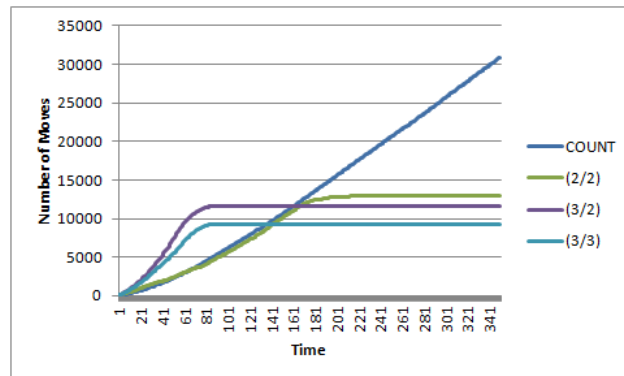


Figure 4.5: Scenario 1: Total Number of Moves of the Sensors for Different Rules in the Case of $R_c = 4$ and $R_s = 1$

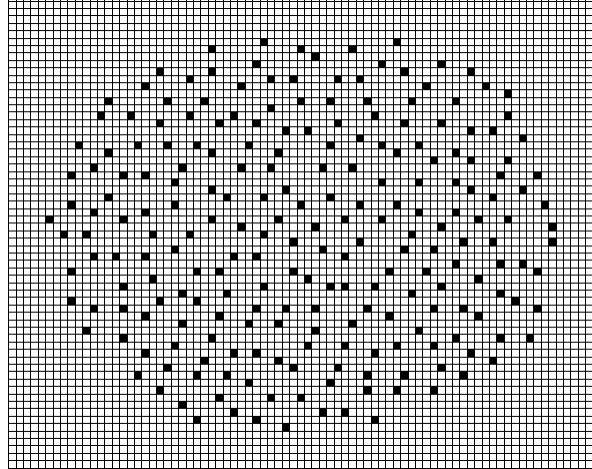


Figure 4.6: Scenario 1: The Positions of the Sensors at Time 100 (3/3) in the Case of $R_c = 4$ and $R_s = 1$ for an Experiment

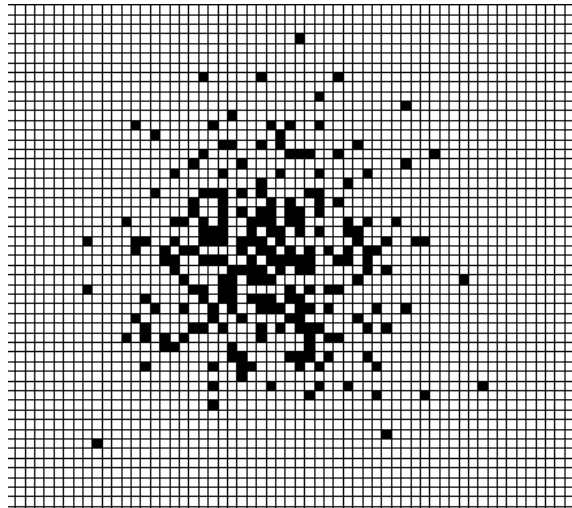


Figure 4.7: Scenario 1: The Positions of the Sensors at Time 100 (COUNT) in the Case of $R_c = 4$ and $R_s = 1$ for an Experiment

sensor is allowed to move beyond these bounds. This type of bound can be helpful if one considers energy constraints on sensor movements. We ran this experiment for 350 time periods and the final coverage is shown in Table 4.1. We include those rules in the table that give us a competitive coverage when compared to the optimal.

| Movement Bound | Rules | Final Coverage |
|----------------|-------|----------------|
| 30 | COUNT | 1379 |
| | 2/2 | 1934 |
| | 3/2 | 1522 |
| | 3/3 | 1517 |
| 50 | COUNT | 1380 |
| | 2/2 | 1438 |
| | 3/2 | 1850 |
| | 3/3 | 1831 |
| 70 | COUNT | 1337 |
| | 2/2 | 1934 |
| | 3/2 | 1988 |
| | 3/3 | 1918 |

Table 4.1: Experimental Results on Different Upper Bounds on the Movements in Scenario 1 in the Case of $R_c = 4$ and $R_s = 1$

The comparison of our algorithm (the best rule among all the rules) and “COUNT” in the case of $R_c = 3$ and $R_s = 1$ is shown in Figure 4.8. A similar comparison in the case of $R_c = 2$ and $R_s = 1$ is shown in Figure 4.9. In both cases, the (2/2) rule performs better than all other rules of our algorithm and the “COUNT” algorithm.

Scenario 2

In this case, we place 100 sensors as a cluster at the north east and the south west corners of the grid, respectively (Figure 4.10). So the optimal solution is 1800. The results of this experiment in the case of $R_c = 4$ and $R_s = 1$ are shown in Figures 4.11

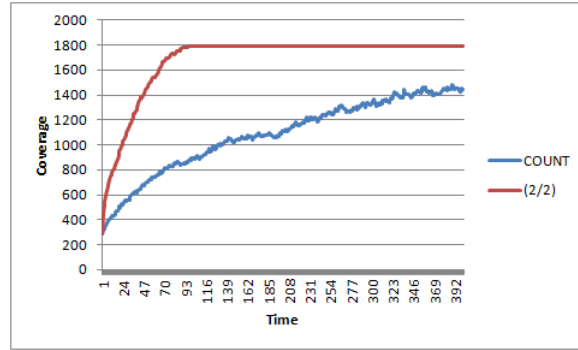


Figure 4.8: Scenario 1: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 3$ and $R_s = 1$

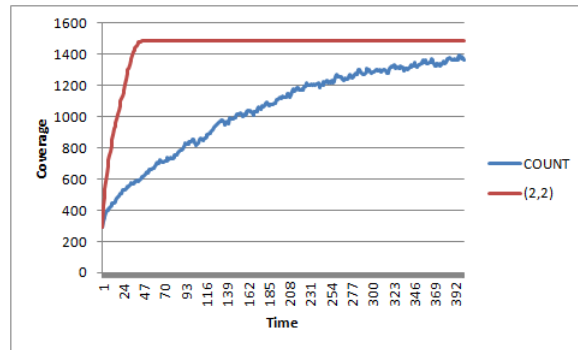


Figure 4.9: Scenario 1: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 2$ and $R_s = 1$

and 4.12 and rules 2/2 and 3/2 give us the optimal solution. Among these rules, rule 2/2 takes more time to reach an optimal configuration. In terms of sensor movements, rule 3/3 is better than the other rules and also gives us a near optimal coverage. The results for different upper bounds for time period 100 are shown in Table 4.2. Time periods and bounds are different than in scenario 1 because these bounds are totally dependent on the scenarios.

Similar comparisons in the case of $(R_c, R_s) = (3, 1)$ and $(R_c, R_s) = (2, 1)$ are shown in Figures 4.13 and 4.14.

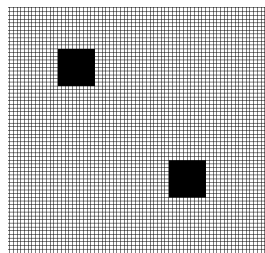


Figure 4.10: Scenario 2

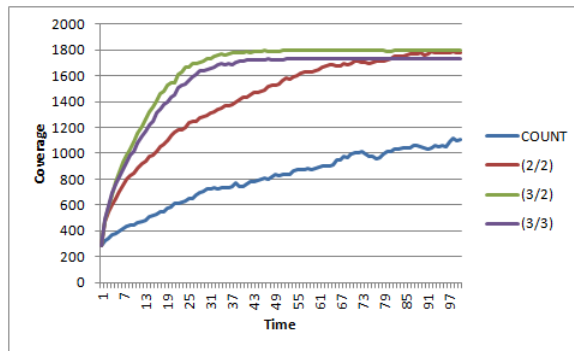


Figure 4.11: Scenario 2: Comparison of the Total Coverage for Different Rules

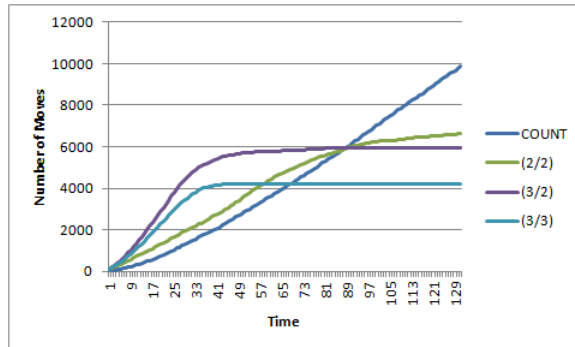


Figure 4.12: Scenario 2: Total Number of Moves of the Sensors for Different Rules

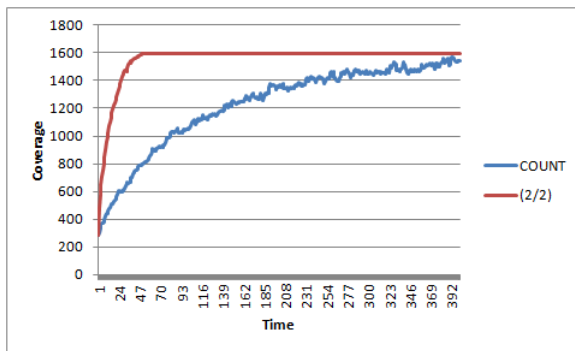


Figure 4.13: Scenario 2: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 3$ and $R_s = 1$

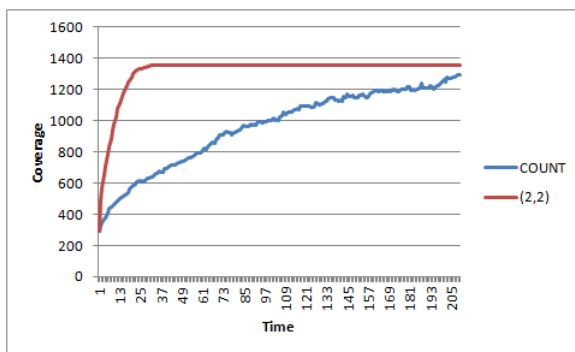


Figure 4.14: Scenario 2: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 2$ and $R_s = 1$

| Movement Bound | Rules | Final Coverage |
|----------------|-------|----------------|
| 20 | COUNT | 1367 |
| | 2/2 | 1368 |
| | 3/2 | 1621 |
| | 3/3 | 1593 |
| 30 | COUNT | 1073 |
| | 2/2 | 1533 |
| | 3/2 | 1741 |
| | 3/3 | 1702 |
| 40 | COUNT | 1060 |
| | 2/2 | 1742 |
| | 3/2 | 1770 |
| | 3/3 | 1724 |

Table 4.2: Experimental Results on Different Upper Bounds on the Movements in Scenario 2 in the Case of $R_c = 4$ and $R_s = 1$

Scenario 3

We place 100 sensors as a cluster at the four corners of the grid, respectively, as shown in Figure 4.15. In this case, the optimal solution is 3600. The results of this experiment are shown in Figures 4.16, 4.17, 4.18, and 4.19. We can find from the simulation that, except for “COUNT” and rule (2/2), all the rules reach a near optimal coverage, but the rule (3/3) reaches it more quickly in terms of time and sensor movements. The experimental results for different movement bounds for time period 80 are shown in Table 4.3.

From all the experiments on three different scenarios, we find that our algorithms give us a very competitive coverage (optimal or near to optimal) quickly in terms of time and number of sensor movements compared to “COUNT” and among these rules, rule 3/3 gives us a competitive coverage with minimum sensor movements.

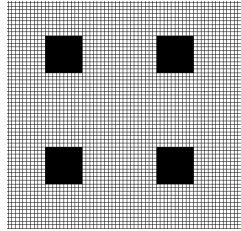


Figure 4.15: Scenario 3

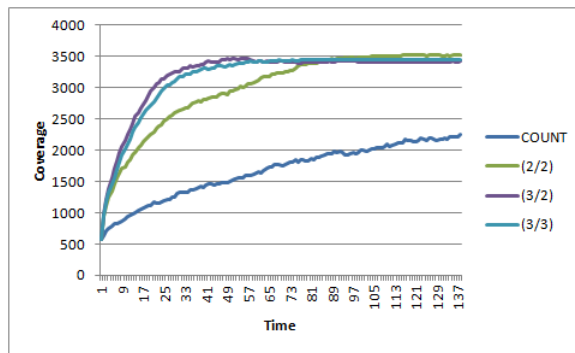


Figure 4.16: Scenario 3: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 4$ and $R_s = 1$

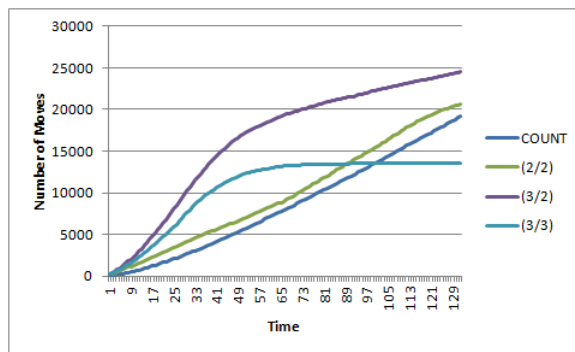


Figure 4.17: Scenario 3: Total Number of Moves of the Sensors for Different Rules in the Case of $R_c = 4$ and $R_s = 1$

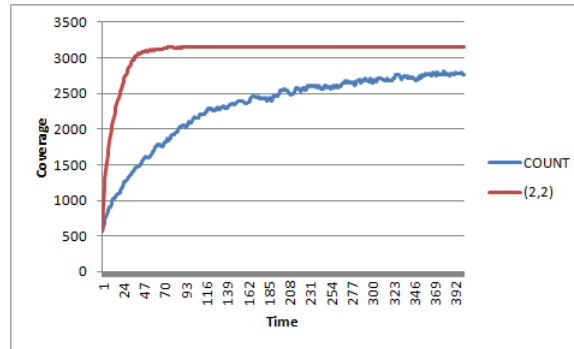


Figure 4.18: Scenario 3: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 3$ and $R_s = 1$

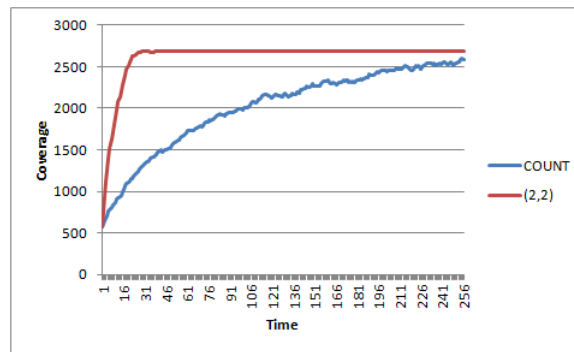


Figure 4.19: Scenario 3: Comparison of the Total Coverage for Different Rules in the Case of $R_c = 2$ and $R_s = 1$

| Movement Bound | Rules | Final Coverage |
|----------------|-------|----------------|
| 10 | COUNT | 1940 |
| | 2/2 | 1931 |
| | 3/2 | 2313 |
| | 3/3 | 2309 |
| 20 | COUNT | 1926 |
| | 2/2 | 2571 |
| | 3/2 | 3083 |
| | 3/3 | 3020 |
| 30 | COUNT | 1990 |
| | 2/2 | 2993 |
| | 3/2 | 3282 |
| | 3/3 | 3200 |

Table 4.3: Experimental Results on Different Upper Bounds on the Movements in Scenario 3 in the case of $R_c = 4$ and $R_s = 1$

4.5 Connectivity Preserving Dispersion Problem

In this variant of the problem we also use 2-dimensional cellular automata to model MWSN and the sensors move in an unbounded 2-dimensional square grid which makes connectivity central. Another difference between this variant and the earlier one is that, unlike the first variant, here one square (or cell) can contain more than one sensor. In one time step, a sensor can move one square in any direction (also diagonally) similar to the earlier variant.

When $R_s \geq \frac{1}{2}R_c$, the sensor deployments that maximize coverage maintaining the connectivity are, essentially, long chains which can be considered “bad” for most applications. For this reason, instead of coverage, our algorithm tries to maximize strongly connected coverage as defined in this section.

The optimal solutions for different (R_c, R_s) pairs are described below. Note that

Lemmas 1 and 2 give bounds for optimal arbitrarily chosen fixed deployments of sensors, as opposed to final configurations produced by our algorithm.

Lemma 1 with $R_c = 3$ and $R_s = 1$, a network with n sensors has optimal coverage $9 * n$.

Proof: In the case of $(R_c, R_s) = (3, 1)$, if we deploy all n sensors, then each sensor can cover a maximum of 9 cells including the cell where it resides, as it has sensing radius 1. Moreover, it is easy to calculate the optimal solution. As the communication radius is 3, each sensor can be placed in such a way so that it is exactly radius 3 away from all of its neighbors while keeping them connected, and each sensor can cover a total of 9 disjoint cells. One such arrangement is shown in Figure 4.20 for 100 sensors with $R_c = 3$. A cell that has a sensor is marked as black. Note that the sensors are not mobile in this configuration. This is the final configuration where the sensors do not need to move any more to improve the coverage.

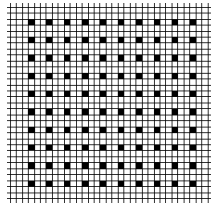


Figure 4.20: Placements of Total 100 Sensors for $R_c = 3$

Lemma 2 If there are n sensors deployed in total, then the optimal global coverage is $9 * n - (n - 1)$ for $(R_c, R_s) = (2, 1)$.

Proof: In the case of sensing radius 1 and communication radius 2, in order to ensure that all the sensors are connected, we can just make a diagonal chain of n sensors where each of the sensors is exactly radius 2 away. But in this case, between any two sensors, there will be one cell that is covered by both of the sensors. So if there are

n sensors, we can get the optimal solution if we make a chain of length n and hence the optimal global solution is $9 * n - (n - 1)$.

Lemma 3 The optimal global coverage for n sensors is $25 * n - (4n - 4)$ for $(R_c, R_s) = (3, 2)$.

Proof: In this case, again we can make a chain of length n to determine the global optimal solution where the distance of the two neighboring sensors in the chain is exactly 3. However, in this case two neighboring sensors share four sensors within their sensing radius. So the global optimal solution is $9 * n - (4n - 4)$.

The optimal solutions for the cases $(2, 1)$ and $(3, 2)$, respectively, are problematic in the sense that they consist of one chain of sensors. In these cases, the distance between the two furthest nodes of the chain is $n - 1$. In sensor networking, this distance (called the hop distance) is important in routing the data among the sensors. To minimize the routing cost of the network, this distance should also be minimized. Another major disadvantage of these solutions is that if any sensor within the chain is dead, then the network will be disconnected. For this reason we measure the performance of the algorithms in terms of strongly connected coverage.

The *strongly connected coverage* of a network (aSCC) is defined as the ratio between the average coverage of the network and the average hop distance of the network. The average coverage of a network (aCOV) is the ratio between the total coverage of the network and the number of sensors of the network and the average hop distance of a network (aHD) consisting of n sensors is

$$\frac{\sum_{m_1 \neq m_2} \text{hd}(m_1, m_2)}{n(n - 1)}.$$

where the hop distance, between m_1 and m_2 , $\text{hd}(m_1, m_2)$, is the length of the shortest

path of sensors connecting m_1 and m_2 .

4.5.1 Algorithms

In this section, we give a high-level description of our algorithm that disperses sensors from an initial configuration while trying to maintain connectivity. In particular, for simplicity, below we talk about the movement of an individual sensor. However, in the general case, one cell may contain more than one sensor and the state of the cell needs to remember the information for each sensor it contains. We consider a scenario where $R_c > R_s$.

The algorithm determines the movement direction of a sensor m_i based on the weighted number of neighbors of m_i in the positive and in the negative x -direction (respectively, y -direction). The weights assigned to neighbors in case of $R_c = 3$ are as follows: The weights for neighbors at distance 1, 2, 3 are 4, 2, 1 respectively. In case of $R_c = 2$, the weights for neighbors at distance 1, 2 are 2, 1 respectively.

In an ideal case, one sensor can have neighbors at distance 3 away in case of $R_c = 3$ and $R_s = 1$ to maximize the coverage while maintaining connectivity. For this reason we assign the weights to be inversely proportional to the distance.

The state representing an individual sensor is a pair (x, y) , $x, y \in \{-1, 0, 1\}$. The state remembers the last move of the sensor. The direction of the movement is stored in the state because, in order to avoid infinite loops, the algorithm preserves the current movement direction. For example, a pair $(0, 1)$ means that in the last time step the sensor did not move in the x -direction and moved upwards along the y -direction. The next movement step of a sensor m_i is determined by the weighted neighborhood of

m_i and the previous movement direction of m_i that is stored in the pair of integers representing m_i . When a cell has more than one sensor, each represented by a pair (x, y) , $x, y \in \{-1, 0, 1\}$, the algorithm computes the potential movement direction for each of these sensors. (The movement direction depends on (x, y) and, thus, may be different for different sensors in the same cell.)

The algorithms are discussed in the following subsections. First we describe how the algorithm determines the direction of the next movement step. Then we describe blocking rules that are used to prevent loss of connectivity and, finally, we describe additional move-back rules that are used when the blocking rules fail.

Local Rules Defining the Movement of Sensors

The algorithms use a parameter (Multiplier), $M \geq 2$ that serves to encourage the sensor to keep moving in the direction of its previous movement step. We define the movement rule as follows.

Suppose a sensor m_i is represented by pair (s_x, s_y) . Suppose that w_1 is the sum of weights of neighbors of m_i in the negative x -direction (to the left of m_i). Suppose that w_2 is the sum of basic weights of neighbors of m_i in the positive x -direction (to the right of m_i). The potential movement of m_i in x -direction depends on the value of s_x , i.e. by remembering the last move, the sensor tries to keep moving in the same direction, unless there is a really good reason to change direction. So the movement of a sensor along the x -direction depends on its neighbors in the positive and negative x -direction and the current value of s_x . If we use a multiplier M , then the decision of movement to the x -direction is determined by a value $x\text{-move}(m_i)$ defined as

- if $s_x = 0$, $x\text{-move}(m_i) = w_2 - w_1$

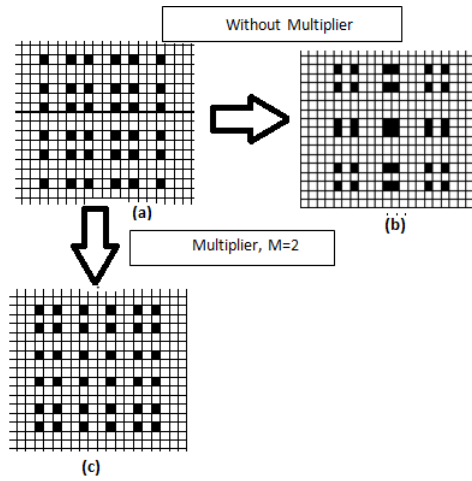


Figure 4.21: An Example of a Cycle and the Usefulness of Multiplication Factor

- if $s_x = -1$, $x\text{-move}(m_i) = (M * w_2) - w_1$
- if $s_x = 1$, $x\text{-move}(m_i) = w_2 - (M * w_1)$

Now, if $x\text{-move}(m_i) = 0$ then the sensor does not move in the x -direction. If $x\text{-move}(m_i) \geq 1$ and $x\text{-move}(m_i) \leq -1$ then the sensor moves to the negative and positive direction, respectively. Movement in the y -direction is determined analogously.

If we do not have multipliers, that is, set $M = 1$, the movement rules defined by the weighted neighborhood of a sensor together with the move back rules (discuss later) can lead to infinite cycles. This is illustrated by the following example (Figure 4.21).

We consider a network with communication radius 3 and the initial configuration consists of sensors in a 6×6 square. After some t time steps the configuration is depicted in 4.21(a) (the cell marked as black contains a sensor) and in the next time step the configuration is as in Figure 4.21(b). Note that in 4.21(a) we have a situation that is discussed later in Figure 4.25 and the connectivity of the network is broken in Figure 4.21(b). Hence the network applies the move back rules (discussed later),

and the resulting configuration is the same as Figure 4.21(a), which means that the network has entered into an infinite cycle. For this reason the algorithm introduces a multiplier $M \geq 2$ that encourages a sensor to keep moving in its previous direction. The sensor (*i.e.*, the state of the cell containing this sensor) remembers the previous movement direction and weights of neighbors in the opposite direction are multiplied with M . Going back to the example depicted in Figure 4.21, the computation step from configuration 4.21(a) using a multiplier, $M = 2$ yields a configuration as in 4.21(c).

Rules for Blocking Movement

We introduce rules that attempt to prevent the network from losing connectivity. Below we consider a movement step in the positive x -direction. The same rules apply to the 3 other directions. If a sensor m_i does not see any neighbors within distance $R_c - 1$ in the negative x -direction, a movement step in positive x -direction is blocked. Consider a cell with n' sensors $m_1, m_2, \dots, m_{n'}$. Each sensor determines independently whether it should move or not. If all the sensors move to a positive x -direction then sensor m_1 checks within distance $R_c - 1$ in the negative x -direction for the connectivity and if there is no sensor within distance $R_c - 1$ then only sensor m_1 will not move to the positive x -direction. In case all sensors in the cell try to move in one of the other three directions, a similar control step is done in the opposite direction.

Additionally, if we have a situation as described in Figure 4.22. We consider a situation where the communication radius R_c is 3. From Figure 4.22, we can see that sensor 3 does not move at the next time step (Figure 4.22(b)) as it does not satisfy the rules of the algorithm in Figure 4.22(a). However, the sensor 2 satisfies the rules and

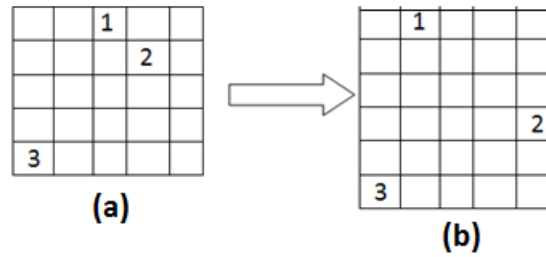


Figure 4.22: Example: Connectivity Breaks. $R_c = 3$

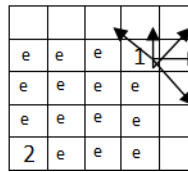


Figure 4.23: Example: Additional Blocking Rule. $R_c = 3$

moves to the positive x and negative y -directions (Figure 4.22(b)) which makes sensor 3 disconnected as it was only connected with sensor 2. So without any movement, a sensor can get disconnected. To solve this problem, we introduce one additional rule. Again considering $R_c = 3$, when we have the same configuration in Figure 4.23 and all positions marked “e” in the figure (common communication range between sensor 1 and 2) are empty, sensor 1 is not allowed to move away from the communication radius of sensor 2 (independently of its weighted neighborhood). This rule applies symmetrically to 3 other orientations.

In case of multiple sensors in a cell, if we have a situation that sensors $m_1, \dots, m_{n'}$ in a cell C_1 see a neighbor diagonally to the south west at distance R_c in cell C_2 and all the cells of the square with corners C_1 and C_2 are empty, then the algorithm prevents sensor m_1 from moving in the positive x -direction (or the positive y -direction). Other sensors in cell m_1 follow the movement rule. This situation, with $R_c = 3$, is illustrated in Figure 4.24. Here the cells marked as “e” are empty.

| | | | |
|-------|---|---|-------|
| e | e | e | C_1 |
| e | e | e | e |
| e | e | e | e |
| C_2 | e | e | e |

Figure 4.24: A situation Where Sensor m_1 in C_1 cannot Move in the Positive x (or y) Direction

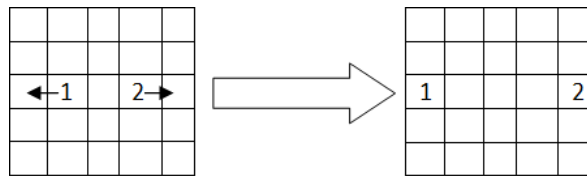


Figure 4.25: Neighbors Break Connectivity for $R_c = 3$

Note that, here it is quite possible that there are other sensors to the north west of C_1 and the original rule for preserving connectivity does not apply. The same rule is used for movement in the other three directions.

Move-Back Rules

The above blocking conditions still do not guarantee the preservation of connectivity because the sensors that are within distance $R_c - 1$ of each other can move to the opposite directions as in Figure 4.25. For this reason we introduce the following “move back” rules. At any given time period t , before moving to the positive x -direction, a sensor s remembers whether or not it has a neighbor in two different quadrants (marked as circles Q_1 and Q_2 in Figure 4.26) in the negative x -direction. At the next time period $t + 1$, if the sensor finds that there is no sensor in one of these quadrants but there was at least one sensor in the same quadrant at period t , then the sensor moves back in the negative x -direction. The same rule applies to 3 other directions.

If we are concerned only about the loss of connectivity, the move back rule can be

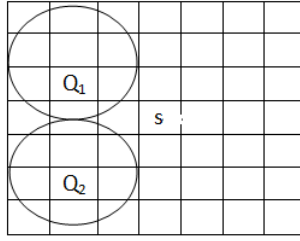


Figure 4.26: Quadrant Move Back Rule with $R_c = 3$

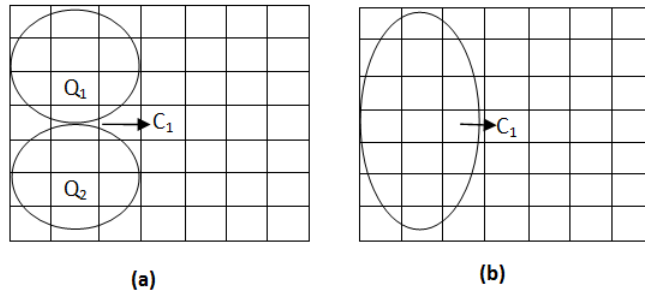


Figure 4.27: Move Back Rules for a Cell with Multiple Sensors with $R_c = 3$. (a) Quadrant Move Back, (b) 180° -move back

simplified by remembering only whether or not there were neighbors in the direction opposite to the current movement, that is, in case of Figure 4.26, quadrants Q_1 and Q_2 can be combined together. We call this simplified rule the 180° -move back rule. The purpose of the quadrant-rules is to prevent the creation of holes in the network. Large holes increase the hop-distance between individual sensors and, thus, worsen the strong connectivity of the network (defined in the next section).

If we have a cell with n' sensors and $x\text{-move}(m_i) > 0$ for all $i = 1 \dots l$, where $l \leq n'$, then in case of 180° -move back rule, only sensor m_1 checks cells for a distance R_c in the negative direction and if it finds that there is no sensor in the negative direction then it moves back in the negative direction. All other sensors move according to the other rules. On the other hand, in the case of quadrant move back, before moving to

a new direction each sensor remembers whether there is any sensor in the quadrants of the opposite direction.

At any period t , m_1 checks the quadrants of the opposite direction and if it finds that there is no sensor in one of the quadrants at that time period but there was at least one in that quadrant at time $t - 1$, then only sensor m_1 moves back in negative x -direction. Other sensors continue with other rules. These rules are applied similarly in the other three directions.

For example, in Figure 4.27, a cell C_1 contains multiple sensors and all came from west then, in case of 180° -move back rule sensor m_1 (one of the sensors in C_1) checks R_c distance in the west and in case of quadrant move back rule it checks Q_1 and Q_2 quadrants at the west.

In a deterministic version of the algorithm, one sensor checks the rules at each time period to decide whether or not it should move from the current location. We also consider the probabilistic versions of the algorithms where at each time period, each sensor verifies the rules with some probability. Though, in a probabilistic version, it takes more time than the deterministic one to reach a final position but we find that the randomized algorithm often gives a better result and the probability of being in a cycle is much less than in the deterministic cases.

4.5.2 Experimental Results of Mobile Dispersion Algorithms

Here we briefly describe the generation of the initial configurations for the experiments used with the algorithm for the sensor dispersal problem. In our experimental set-up, we initially distribute all sensors densely in a small area. For example, in the case of 50 sensors, we consider a 5×5 area in the middle of the grid and distribute the sensors

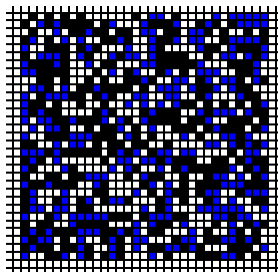
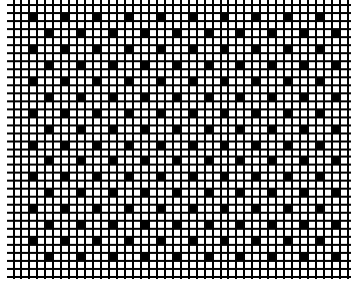


Figure 4.28: A Random Initial Configuration for 1000 sensors

randomly in this area. Using an uniform random distribution we randomly pick a location from this area. A location can be chosen a maximum of three times (*i.e.* a cell can have a maximum of three sensors initially). We continue picking locations in this manner until all the sensors (total 50 in this case) are deployed. Similarly, in the case of 1000 sensors we consider a 30×30 block in the middle of the grid. In all the cases, we also specify that each cell of an area can have a maximum of 3 sensors. One such initial configuration with 100 sensors is shown in Figure 4.28. For each experiment, the algorithms were run 20 times on random initial configurations. A black cell of the grid contains one sensor while a blue cell contains more than one sensor.

We study the sensor dispersal problem experimentally for cases where (R_c, R_s) is $(3, 2)$, $(3, 1)$ or $(2, 1)$. The general idea is that the sensors are initially deployed in a dense and random “clump”, and the local algorithm should disperse the sensors to maximize coverage while, at the same time, maintaining connectivity of the network. In this section, we give an overview of the results for the different algorithms and explain the reasoning of adding various features in the algorithms.

We compare the results of the algorithms with two benchmark configurations, *square* (similar to Figure 4.20) and *star* (Figure 4.29). In the square benchmark n sensors

Figure 4.29: Star Benchmark with $R_c = 3$

are placed in a square formation and the distance between adjacent sensors is R_c , or a “close to square” formation when n is not a perfect square. The star benchmark is obtained from the square benchmark by omitting every second sensor, and alternating the positions on consecutive rows.

When we compare the strongly connected coverages, the star benchmark gives us the better result compared to the square benchmark

Note that, except in the case, $R_c = 3$, $R_s = 1$, the square formation does not provide optimal total coverage, because the area sensed by two adjacent sensors overlaps when the sensors are within the communication radius of each other. However, the square formation gives reasonably good strongly connected coverage with respect to the strongly connected coverage measure.

When we apply our deterministic algorithm with the 180° -move back rule for an initial deployment as Figure 4.3 we always get the square formation for $M = 6$. Thus we get the optimal solution for the $R_c = 3$, $R_s = 1$ case.

However, when we apply the algorithm for different random initial configurations, the results are not very impressive in terms of strongly connected coverage. A possible reason why the results for the deterministic algorithm with the 180° -move back rule

| <i>Sensors</i> | (R_c, R_s) | $M = 2$ | $M = 3$ | $M = 4$ | $M = 5$ | $M = 6$ |
|----------------|--------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| | | aSCC | aSCC | aSCC | aSCC | aSCC |
| | | aCOV / aHD | aCOV / aHD | aCOV / aHD | aCOV / aHD | aCOV / aHD |
| 50 | (3, 1) | 1.77 8.78/4.96 | 1.87 8.98/4.80 | 1.65 8.86/5.34 | 1.82 8.96/4.93 | 1.84 8.96/4.84 |
| 50 | (3, 2) | 3.01 14.94/4.96 | 3.19 15.30/4.80 | 2.94 15.70/5.34 | 3.14 15.50/4.93 | 3.26 15.82/4.84 |
| 50 | (2, 1) | 1.34 6.30/4.68 | 1.27 6.00/4.70 | 1.32 6.02/4.54 | 1.07 6.16/5.77 | 1.28 6.46/5.01 |
| 100 | (3, 1) | 1.33 8.84/6.63 | 1.38 8.59/6.19 | 1.30 8.51/6.50 | 1.34 8.79/6.56 | 1.32 8.81/6.62 |
| 100 | (3, 2) | 2.18 14.51/6.63 | 2.25 13.94/6.19 | 2.14 13.96/6.50 | 2.21 14.51/6.56 | 2.21 14.70/6.62 |
| 100 | (2, 1) | 0.89 5.74/6.48 | 0.93 5.92/6.35 | 0.88 5.89/6.68 | 0.89 5.90/6.67 | 0.91 5.92/6.45 |
| 200 | (3, 1) | 0.98 8.91/9.01 | 0.97 8.83/9.13 | 0.97 8.84/9.14 | 0.97 8.80/9.02 | 0.97 8.82/9.06 |
| 200 | (3, 2) | 1.53 13.82/9.01 | 1.54 14.05/9.13 | 1.53 13.85/9.14 | 1.53 13.81/9.02 | 1.50 13.56/9.06 |
| 200 | (2, 1) | 0.66 5.70/8.58 | 0.66 5.63/8.57 | 0.66 5.68/8.64 | 0.65 5.64/8.62 | 0.64 5.76/9.01 |
| 300 | (3, 1) | 0.82 8.88/10.88 | 0.81 8.84/10.93 | 0.78 8.76/11.19 | 0.80 8.80/10.99 | 0.83 8.89/10.76 |
| 300 | (3, 2) | 1.24 13.45/10.88 | 1.25 13.62/10.93 | 1.24 13.84/11.19 | 1.24 13.60/10.99 | 1.24 13.34/10.76 |
| 300 | (2, 1) | 0.53 5.55/10.41 | 0.54 5.56/10.31 | 0.53 5.51/10.44 | 0.54 5.63/10.41 | 0.54 5.66/10.51 |

Table 4.4: Comparison Results for Different Multiplication Factors with 180°-moveback rule

a given sensor m_i with probability p follows the rules of the algorithm to determine whether or not s should move). Another modification that tries to prevent the creation of holes is the quadrant rules. As we expect better results from the quadrant move back rules in terms of average hop distance we apply this variant of the algorithm on 20 different random initial configurations. We find that this does not give competitive results in deterministic cases because some cycles occur in the solutions quite early. Though we do not get competitive results for the deterministic quadrant move back version of the algorithm, we get better results for probabilistic quadrant move back version compared to the other variants of the algorithm including the probabilistic 180°-move back rule.

The probabilistic variant of the algorithm and the quadrant rules are introduced to reduce holes in the final configuration and, thus, to improve strongly connected coverage. A typical final configuration is illustrated in Figure 4.31. The configuration of Figure 4.31(a) has smaller holes than configurations produced by the earlier algorithm Figure 4.31(b). The reason why even the quadrant algorithm cannot completely prevent the creation of holes is illustrated by Figure 4.32. In this example, sensor 2 satisfies the rules and moves north west in the next time period. Sensor 7 loses the neighbors in that quadrant in the next time period as sensor 2 will not move back because it will still find sensor 4 as a neighbor in the opposite quadrant.

We use different numbers of sensors in our experiments and for each case we consider 20 different initial configurations for a given probability value p and we take the average of these 20 experiments. We find that the probabilistic quadrant move back algorithm gives us better results for $p = 0.2$ compared to all other variants of the algorithms.

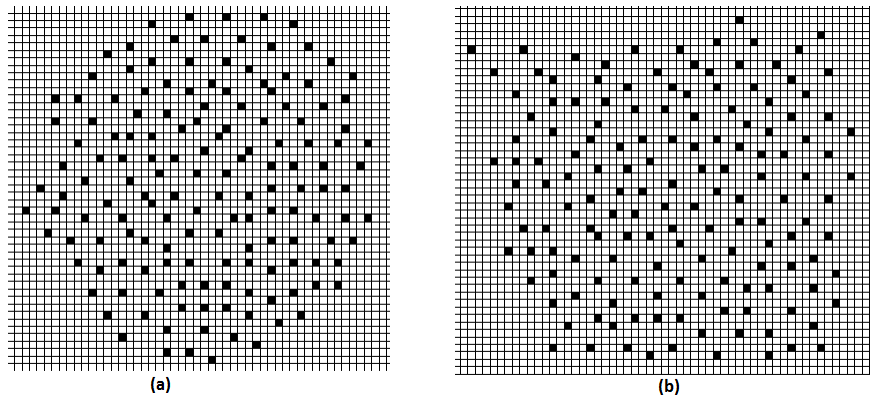


Figure 4.31: Comparison of the Final Configurations: (a) Probabilistic Quadrant Move Back, (b) Probabilistic 180-degree Move Back rule

| | | | | | |
|---|---|---|---|---|---|
| 1 | | | | | |
| | | 2 | | | |
| 3 | | | | 4 | 5 |
| | | | | 6 | |
| | | | 7 | | |
| | 8 | | | | |
| | | | 9 | | |

Figure 4.32: Example of a Hole in Quadrant Move Back

Figure 4.33 shows the average coverage of a network for different number of sensors for two probabilistic algorithms with $R_c = 3$ and $R_s = 2$. We find that in terms of average coverage, the probabilistic 180-degree-move back rule gives us better results, but in terms of strongly connected coverage, the Quadrant Move back performs better because of the lower average hop distance of the network (Figure 4.34). Note that to show the comparison we consider the best result from both algorithms ($p = 0.2$). We show the strongly connected coverage of a network for both probabilistic algorithms with (R_c, R_s) pairs $(2, 1)$ and $(3, 1)$ in Figures 4.35 and 4.36.

In fact, both versions of our probabilistic algorithm give on average slightly better strongly connected connectivity than the square benchmark configuration for the $(3, 2)$ and $(2, 1)$ pairs. Figure 4.37 illustrates final configurations with 1000 sensors

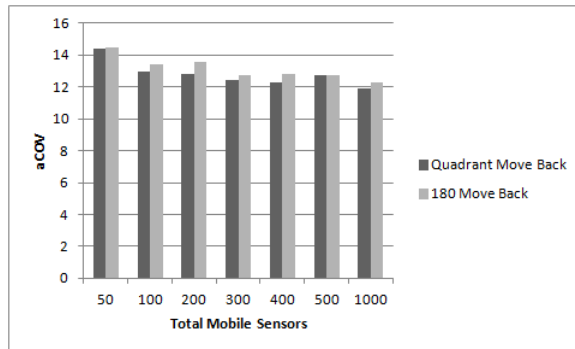


Figure 4.33: Comparison of aCOV Among Two Probabilistic Algorithms for Different Number of Mobile Sensors With $R_c = 3$ and $R_s = 2$

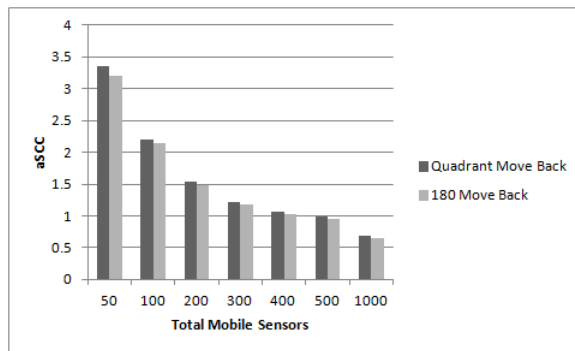


Figure 4.34: Comparison of aSCC Among Two Probabilistic Algorithms for Different Number of Mobile Sensors With $R_c = 3$ and $R_s = 2$

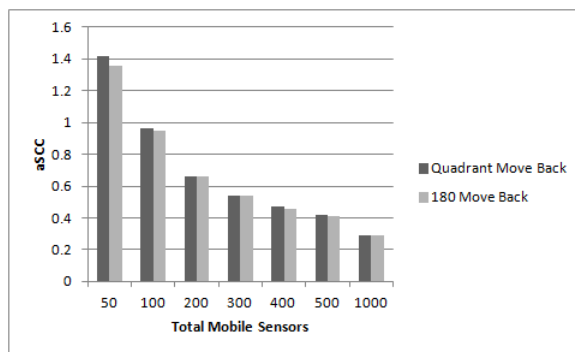


Figure 4.35: Comparison of aSCC for Different Number of Mobile Sensors with $R_c = 2$ and $R_s = 1$

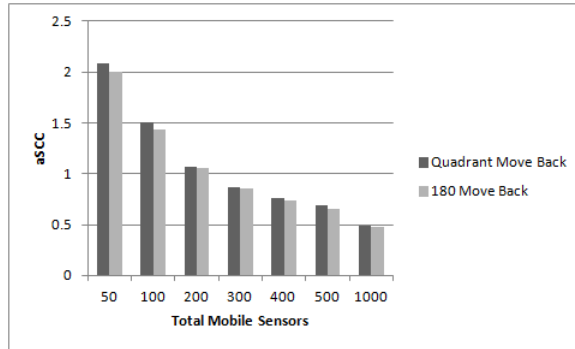


Figure 4.36: Comparison of aSCC for Different Number of Mobile Sensors with $R_c = 3$ and $R_s = 1$

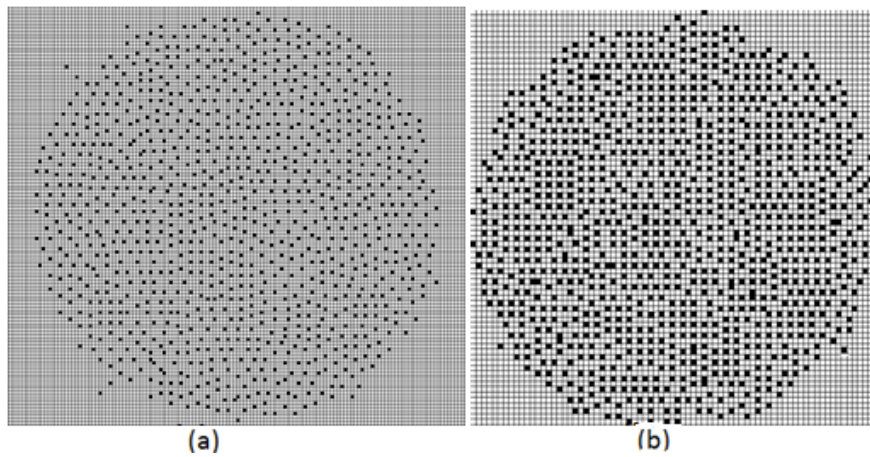


Figure 4.37: Example: Final Configurations With 1000 Sensors for a) (3,2) and b) (2,1)

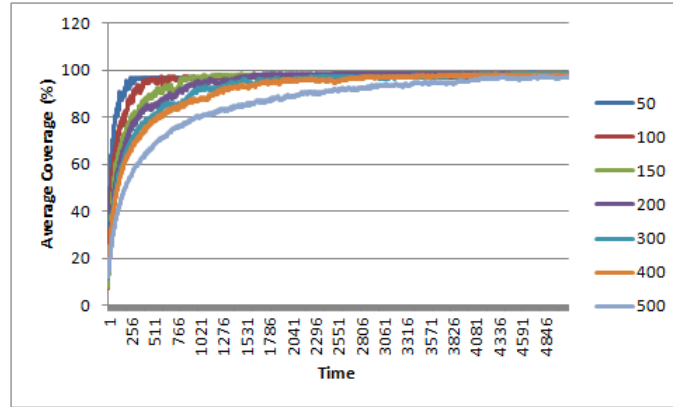


Figure 4.38: Average Coverage Vs Time period (Probabilistic Quadrant Move Back)

where the result is, roughly, 4% better than in the benchmark configuration. The explanation is that when the ratio R_c/R_s becomes smaller, the increase in hop-distance caused by holes in the network is more than compensated for by less overlap between sensing radii of adjacent sensors. With a larger sensing radius the algorithm should try to disperse the sensors more aggressively.

We present the total coverage of the network in percentage for both probabilistic quadrant move back and probabilistic 180° -move back rule with $p = 0.2$ for different numbers of sensors (50, 100, 200, 300, 400, 500 and 1000) with $R_c = 3$ and $R_s = 1$ in Figures 4.38 and 4.39. The X axis of the figures represent the time period and the Y axis represent the total coverage of the network in percentage. These results show the average of 20 experiments. From these figures we find that both algorithms give us a very competitive average quite quickly.

Finally, we compare the best results of the probabilistic algorithm (quadrant move back rule with $p = 0.2$) with benchmark configurations in Figures 4.40 and 4.41. The Y axis represents the ratio (raSCC) of the average strongly connected coverage for the results produced by the algorithm and for the benchmark square configurations,

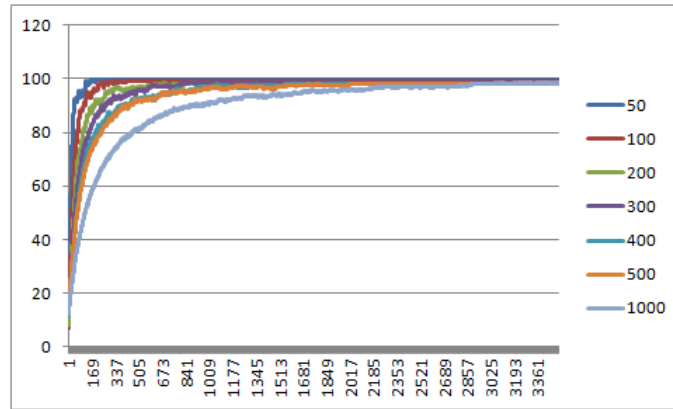


Figure 4.39: Average Coverage Vs Time period (Probabilistic 180°-move back)

respectively. Note that in the case of both (3, 2) and (2, 1), the strongly connected coverage of the network produced by the probabilistic algorithm is better than the square benchmark because in this case, though aHD (average hop distance) is better in the square benchmark, but the latter gives less average coverage than the probabilistic algorithm. However, the star benchmark is better than our algorithms and our algorithms give a competitive results with an increase in the number of sensors.

4.6 Summary

In this chapter of the thesis we proposed a set of probabilistic and deterministic algorithms for two variants of the mobile dispersion problem. First, we proposed an algorithm for maximizing the coverage while in the second variant we also considered connectivity preservation constraints. We performed extensive simulations for all the algorithms. Our cellular automaton based algorithms for the first variant outperformed the existing similar local algorithm. Our probabilistic algorithms for the latter variant performed better than the deterministic algorithms. We also compared

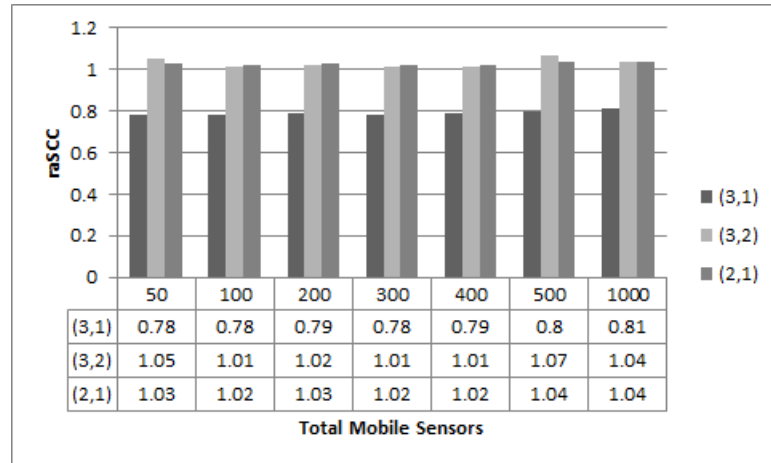


Figure 4.40: raSCC With Respect to Square Benchmark for the Probabilistic with Quadrant Move Back Algorithm for Different (R_c, R_s) Pairs

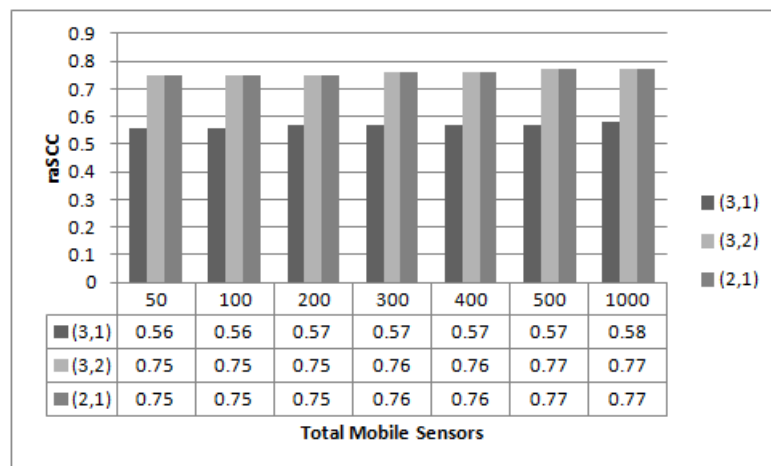


Figure 4.41: raSCC With Respect to Star Benchmark for the Probabilistic With Quadrant Move Back Algorithm for Different (R_c, R_s) pairs

our algorithms with two benchmarks and found that our algorithms performed better than the square benchmark and gave competitive results with respect to the star benchmark in terms of the strongly connected coverage metric.

Chapter 5

Synchronous Gathering of Mobile Sensors

5.1 Introduction

Normally a large number of sensors are deployed in the environment to sense the data and they can disperse to increase the coverage of the network as we saw in chapter 4. In an alternative scenario, the sensors need to autonomously move to a single location where they can all communicate with each other directly or where they can be collected for later use. For example, after a certain period of time at the field, the sensors might start losing their energy and want to be back into a position where they can be collected all together.

An analogous problem has been studied in robotics [12, 43] under the name of *synchronous gathering problem*. A local algorithm proposed in [12] is considerably more complex than our cellular automaton based algorithm. In this algorithm, each sensor

determines the smallest enclosing circle that contains all of its neighbors and moves towards the center of that circle. Computing this circle needs a higher number of states compared to our local algorithm. Recently Degener *et. al.* [43] prove that the running time taken by a sensor ($O(n^2)$) is tight. A similar problem is considered for the asynchronous case in [49], where a sensor can see what are the movements of its neighbors at a given time period t . We discuss this algorithm later in this chapter when we discuss and compare it with our algorithm.

To the best of our knowledge, we propose the first CA based algorithm for the synchronous gathering problem. In this case at a time period t , a sensor M does not know what would be the movement of its neighbors at that time period t . In our algorithm the sensors do not store any previous information and use very limited local information at each time period to find the possible future locations at the next time period. We also show that our algorithm is fast and converges in a finite amount of time and the network remains connected while the sensors move.

This chapter is organized as: section 5.2 defines the problem formally. The algorithm of the problem is described in section 5.3. We analyze our algorithm in section 5.4. The details of the simulations and the results of the algorithm is presented in 5.5. Finally, we present the summary of this chapter in 5.6

5.2 Problem Statement

A set of n mobile sensors is deployed in a two dimensional grid network. They are all connected but sparsely deployed. The goal of the algorithm is to gather all sensors into one cell.

Note that since each sensor knows only its local neighborhood, it is clear that any successful algorithm for the problem has to maintain connectivity of the network during the intermediate stages of the gathering process. If during the execution of the algorithm the network would split into parts D_1 and D_2 that have no connection between them, a further application of the algorithm would gather sensors of D_1 and D_2 to two different cells, respectively.

5.3 Algorithm

We use a two dimensional cellular automaton grid to simulate the movements of the sensors of the network. If any cell of the grid contains at least one sensor then the state of the cell is 1; otherwise the state of a cell is 0. Note that when during the execution of the algorithm more than one sensor enters the same cell, during the rest of the execution the movement of this set of sensors will be identical. Hence the state of a cell needs to “remember” only whether or not the cell has at least one sensor. We consider a radius R_c neighborhood for each cell where R_c is the communication radius of the sensors.

Now we describe the movement rules. Each sensor checks the following rules:

- The sensor checks whether it sees any other sensor in its neighborhood in the positive (respectively, negative) vertical direction. If it finds that both of the directions have at least one cell that is in state 1 then the sensor does not move.
- If both of the directions have only cells that are in state 0 then the sensor does not move.

- If the sensor finds one direction that has at least one cell that is in state 1 and the other direction has no such cell then the sensor moves to that direction that has cell in state 1

The same rules apply to the horizontal axis.

Basically, at each time period if a sensor finds one direction empty (no sensor) and other direction non-empty (has at least one sensor), then the sensor moves towards the non-empty direction.

When we apply these rules together we find that in some cases, the sensors in close by cells enter into a cycle. If we make a smallest enclosing rectangle that contains these cells then we get either a 2×2 or a 1×2 rectangle.

To prevent infinite cycles occurring inside a 2×2 square the following rules are applied to a mobile sensor M *only when all the sensors that M sees fit inside a 2×2 square.*

The sensor M moves only in following cases:

- i If M sees a sensor directly to the right and no sensor below, then M moves right. That is, we have a situation

$$\begin{array}{cc} M & M' \\ 0 & 0 \end{array}$$

where M' is a cell with another sensor and 0's denote empty cells.

- ii If M sees another sensor directly below and no sensor to the right or diagonally towards south-east, then M moves down.
- iii If M sees another sensor diagonally toward south-east then M moves diagonally south-east.

Note that the above rules are applied only when all sensors in the neighborhood of M fit inside a 2×2 square. Thus, for example, in case (iii) it is required that the cells toward up, left and diagonally north-west from M must be empty.

5.4 Analysis

Theorem 1: The algorithm never breaks the connectivity.

Proof: At any given period of time t , if a network is connected then at time $t + 1$, it will also be connected, because each sensor always moves away from the empty direction and moves towards its neighbors for each of the axes. As there is no sensor in the empty region there is no possibility of breaking connectivity.

Lemma 1: If the sensors gather into a 1×2 square at time t , then they can gather into one cell at time $t + 1$.

Proof: In this case, only the following two cases (Figure 5.1) can occur. Here cells 1 and 2 both have at least one sensor at time t and in both cases sensors that are in cell 1 move to cell 2 at time $t + 1$ using the algorithm described above.

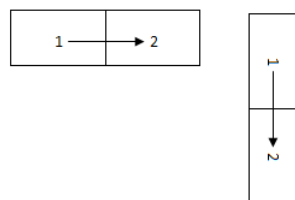


Figure 5.1: An Example of the Final Step of the Algorithm

Lemma 2: If the sensors gather into a 2×2 square at time t , then they gather into a 1×2 square at time $t + 1$.

Proof: If sensors merged into a 2×2 square at time t , then at time $t + 1$, the sensors that are in the left most cells move horizontally towards the right most sensors according to the rules of the algorithm and the sensors converged into a 1×2 square.

Theorem 2: All the sensors gather into a cell in a $O(d^2 * n)$ time where n is the number of sensors and they are initially deployed inside a square where d is the length of an edge.

Proof: At any given period of time t , we can make a smallest enclosing rectangle in a $2D$ grid that contains all the sensors.

Now at time t , at least one sensor from each side of the rectangle (that are the closest to the boundary) will move in because they find an empty region in the other directions. Now assuming one side of the rectangle is larger than two, either we can make a smaller enclosing rectangle at time period $t + 1$ or the number of sensors in the boundary of the enclosing rectangle at time $t + 1$ is the smaller. Moreover, no sensor goes back to the previous position according to the rules of the algorithm. For this reason, all the sensors can gather into a 2×2 or 1×2 square in $O(d^2 * n)$ where d is the grid size and n is the total mobile sensors.

Theorems 1 and 2 imply that the algorithm leads an arbitrary connected network to converge into one cell in a finite amount of time.

5.5 Simulations and Results

Though to the best of our knowledge, we present the first cellular automaton based local algorithm for this problem, we can consider other local algorithm presented in [49] for an asynchronous version of the problem for the sake of comparison. We

consider the algorithm for the synchronous case modeling it as a cellular automaton model. The cells, state and neighborhood description in this algorithm are the same as in our algorithm presented in the previous section. The rules of movement of each sensor are as follows [49]:

- If a sensor finds at least one sensor to its left or above on its vertical axis, it does not move.
- If a sensor finds at least one sensor only below on its vertical axis, it moves down towards the nearest sensor.
- If a sensor finds at least one sensor only to its right, it moves horizontally towards the vertical axis of the nearest sensor.
- If a sensor finds at least one sensor both below on its axis and on its right, it makes a diagonal moves towards right and below.

When we model the algorithm for an asynchronous case, we change the last rule of the algorithm because if we apply this rule as it is then it can cause the network to be disconnected. One such example is shown in Figure 5.2.

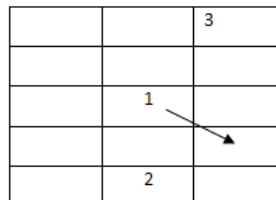


Figure 5.2: An Example Where Connectivity Breaks

In Figure 5.2, the communication radius of each sensor is 2. Now according to the rules of the algorithm, only sensor 1 is allowed to move at time t and if it moves

diagonally right and down then it breaks connectivity with sensor 3 at time $t + 1$. It breaks connectivity with 2 if it moves diagonally right and up.

To overcome this problem, we changed the last rule to: if a sensor finds at least one sensor both below on its axis and on its right then it moves horizontally towards the vertical axis of the nearest sensor.

Note that the original asynchronous robotics algorithm [49] uses a more complicated method to calculate the precise direction of the horizontal move and the precise selection of the movement direction cannot be implemented on a cellular grid. This is the reason we have modified the last rule describing the horizontal movement step.

Note that if we apply this algorithm then it gathers all the sensors to the bottom right corner of the initial enclosing rectangle that is drawn from the initial positions of the sensors whereas our algorithm gathers all the sensors close to the center of this rectangle.

Other main differences between these two algorithms are:

- Because in the algorithm of [49] sensors move from left and top parts of the network to the right and bottom parts, that algorithm takes more time to converge compared to our algorithm. In our algorithm, sensors from all the parts move towards the center.
- The maximum and average number of movements required by a sensor in the algorithm of [49] are more than the respective numbers required by our algorithm.

Though it is expected from the above explanation that our algorithm performs better than the algorithm of [49] in terms of the running time and other metrics (maximum

and average number of movements by a sensor.), we also implement and compare the algorithms for different initial configurations. In our simulations, we consider the communication radius (R_c) of a sensor to be 2.

We compare the algorithms for various numbers of sensors and find that our algorithm converges faster than [49] and in our algorithm the average ($AVGD$) and the maximum number of movements ($MAXD$) by a sensor are each one half of the corresponding values in [49].

Consider an initial configuration where d sensors are deployed in a horizontal (or vertical) line. In our algorithm it takes $\frac{d}{2}$ time periods for all the sensors to reach the middle point, whereas in [49], it takes d time steps to reach the right most position of the line. Similarly, $AVGD$ and $MAXD$ of these configurations are twice in [49] compared to our algorithm.

Now we compare the algorithms for some square formations as in Figure 5.3 and compare the results for the different number of sensors in Figures 5.4, 5.5, 5.6. In these figures we denote our algorithm and the algorithm of [49] as *Algorithm1* and *Algorithm2* respectively.

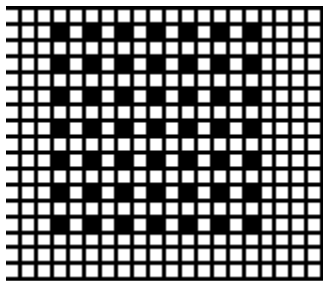


Figure 5.3: A Square Configuration

Figure 5 shows that our algorithm takes much less time than [49] (about half) to gather all the sensors in a point. We compare $AVGD$ and $MAXD$ in Figures 5.5

and 5.6 and find that in both cases our algorithm performs better.

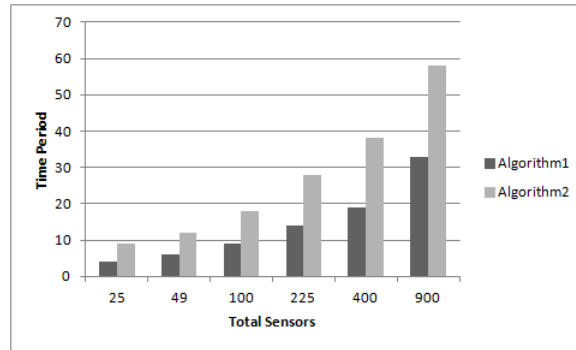


Figure 5.4: Comparison of Time Periods Taken by the Algorithms

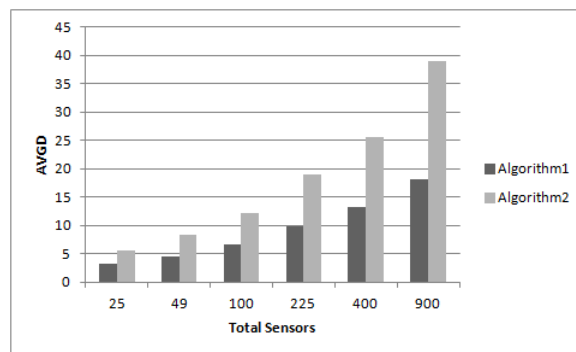


Figure 5.5: Comparison of AVGD

We also compare the algorithms for different random connected networks and find that our algorithm performs better in all cases in terms of all the metrics. We consider the final outputs of the algorithms proposed in [29] as the initial configurations for our comparisons. The algorithms proposed in [29] dispersed the mobile sensors from a dense initial configuration to maximize the coverage while maintaining the connectivity. The final configurations returned by the algorithms are sparse and connected. One such configuration is shown in Figure 5.7.

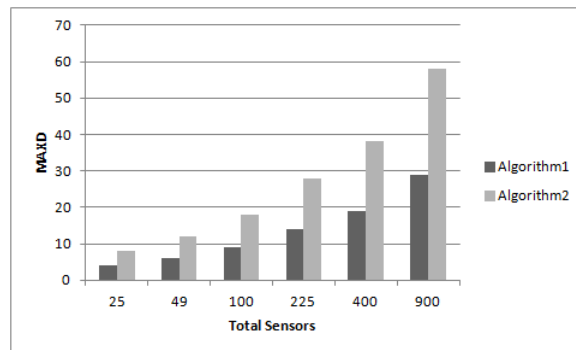
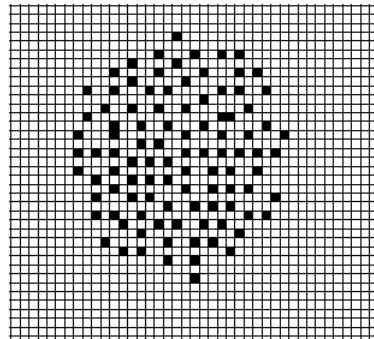


Figure 5.6: Comparison of MAXD

Figure 5.7: An Example of a Random Configuration of the Mobile Sensors with $R_c = 2$

We consider 10 different configurations for different numbers of sensors (50, 100, 200, 400, 500 and 1000) and take the average results. The results are shown in Figures 5.8, 5.9, and 5.10. It can be noted that, on average, the speed-up of our algorithm compared to the algorithm of [49] is greater than a factor of two.

From all these experiments we find that our algorithm performs much better in terms of the metrics considered to compare.

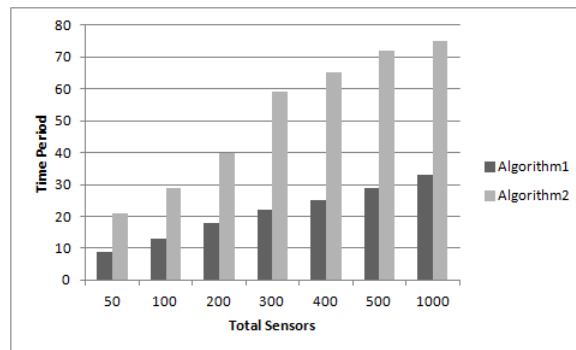


Figure 5.8: Comparison of Time Periods Taken by the Algorithms

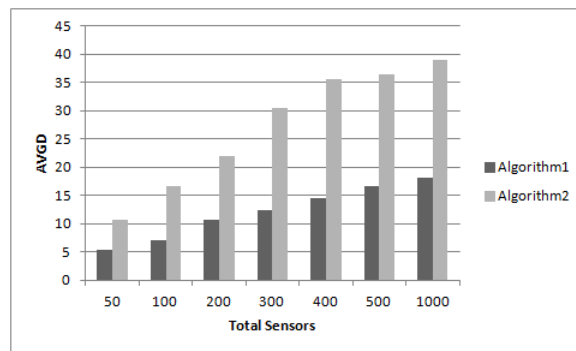


Figure 5.9: Comparison of AVGD

5.6 Summary

We proposed a deterministic algorithm for synchronous gathering problem where a set of dispersed mobile sensors needs to gather in a single location. We proved that our algorithm converged in a finite amount of time. We compared our algorithm with another modified algorithm and found that our algorithm performed better than the earlier algorithm.

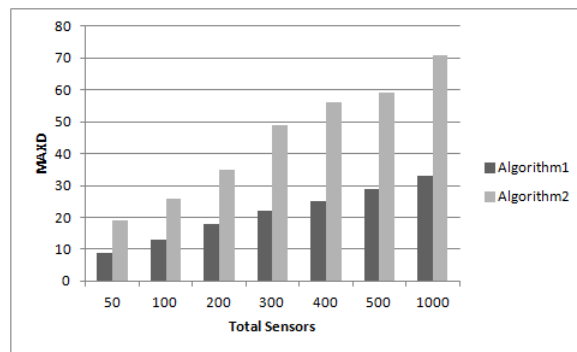


Figure 5.10: Comparison of MAXD

Chapter 6

Minimizing Chain Length Problem

6.1 Introduction

In this chapter we consider another optimization problem related to the movement of mobile sensors in a mobile sensor network, called *the minimizing chain length problem*. We are given a chain of n sensors. All sensors except the two end points of the chain are connected with their two designated neighbors, right and left. The two end points are fixed and have one designated neighbor. Other sensors can move autonomously. At an initial configuration of the network, the chain can be windy and even overlap with itself. Our goal is to move the sensors so that they can come as close as possible to the direct line between the two end points and while the sensors move the network should remain connected. Each sensor can only use its local information (*i.e.*, it only knows the positions of its designated neighbors).

In different applications of MWSN, sensors can be placed as a windy chain and each sensor works as a router. Consider one end point of such a chain is a base station.

Now, if one mobile sink starts moving from another end point to visit each and every router till it reaches the base station at the other end then it might need to travel a long distance. To overcome this problem, each router can move by themselves towards the straight line between the two end points of the chain so that it takes much less time for the explorer to visit them. Moreover, a solution of this problem can be useful to minimize the energy needed to send data from each of the routers (sensor) [46, 65].

There are some local algorithms proposed for this problem [54, 46]. However, the solutions that are proposed for this problem are quite complex if we implement the algorithms in cellular automata. In most of the cases, each sensor calculates the midpoint of its two neighbors and tries to move towards that point [46] which needs much more states compared to our cellular automaton based algorithm.

To the best of our knowledge, we propose the first cellular automaton based algorithms (deterministic and probabilistic) for this problem. Our algorithms give near to optimal solutions and the deterministic algorithm is faster than the probabilistic one.

The organization of this chapter of the thesis is: section 6.2 defines the problem. We present the algorithms of the minimizing chain length problem in section 6.3. The simulations and the results of these algorithms are presented in section 6.4. The summary of this chapter is presented in section 6.5

6.2 Problem Definition

We are given a chain of n sensors, m_1, m_2, \dots, m_n in a two dimensional plane. For each $1 < i < n$, the sensors m_{i-1} and m_{i+1} are within the communication radius of m_i and, furthermore, they are distinguished as the left and right neighbor, respectively

of m_i in the chain. (In the next subsection we will elaborate on how the information distinguishing the left and right neighbors is encoded in the states of the CA.). Each sensor except m_1 and m_n is allowed to move in any direction while m_1 and m_n can not move. Our main goal is to move the sensors as close as possible to the shortest line between m_1 and m_n while not breaking the connectivity.

One such initial chain configuration is shown in Figure 6.1. Sensors are represented as circles and the two end points of the chain are placed within a rectangle in the figure.

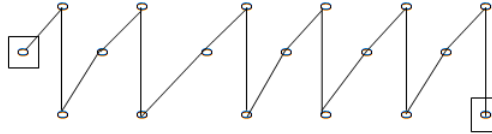


Figure 6.1: Scenario 1: A Chain Configuration

6.3 Algorithms

We represent the state of a sensor as a 6-tuple $(left_x, left_y, right_x, right_y, move_x, move_y)$ where $left_x, left_y, right_x, right_y \in \{-R_c, -R_c + 1, \dots, R_c - 1, R_c\}$, and $move_x, move_y \in \{-1, 0, 1\}$. Here R_c is the communication radius of the sensors.

Here $left_x$ and $left_y$ represent the distance between the sensor and its left neighbor in both the x and y directions respectively before the previous move and $right_x, right_y$ are the x and y distance to the right neighbor.

The components $move_x$ and $move_y$ give the directions of the previous movement step of a sensor.

Now we define the movement rules for the sensors. The movement step consists of three phases. In the first phase each pair of designated neighbors that have moved to the same cell, adjust their left and right neighbors. In the second phase the sensor actually moves (according to rules described below) and in the third phase the sensor updates the left/right values. In the following description, by a neighbor of a sensor we always mean its designated left or right neighbor.

At each time period, each sensor first checks the positions of its designated neighbors. If it finds that the sensor itself and its left neighbor both are in the same cell, that is, in the state of the cell $\text{left}_x = \text{left}_y = 0$, then the sensor makes the left neighbor of its designated left neighbor as its own designated left neighbor. A similar rule is applied in the case of the right designated neighbor. Note that once two sensors that are designated neighbors of each other occupy the same cell, the algorithm moves these sensors identically and hence we can identify their left or right neighbors, respectively.

The sensor then checks the positive and negative directions of its vertical axis:

- If it finds one of its neighbors is in the positive direction and another is in the negative direction, then the sensor does not move.
- If it finds that both of its neighbors are in the same direction, it moves one cell towards that direction.
- If it finds that one of its neighbors is in the positive vertical direction and the other is vertically on the same level as the current sensor, then it moves towards the positive direction. A similar rule applies to the negative direction.

Similar rules apply to the horizontal axis.

Note that a sensor M “knows” the location of its distinguished left and right neighbors at time t , but since the sensors are indistinguishable from each other the algorithm needs also the components giving the last movement directions to guarantee that M knows which sensors are its distinguished neighbors at time $t + 1$. This is done as follows.

Assume that after a movement step in phase 2, sensor M is represented by $(left_x, left_y, right_x, right_y, move_x, move_y)$ and M' is represented by $(left_{x'}, left_{y'}, right_{x'}, right_{y'}, move_{x'}, move_{y'})$.

Now M' is the right neighbor of M if the values

$$right_x - move_x + move_{x'} \tag{6.1}$$

and

$$left_{x'} - move_{x'} + move_x \tag{6.2}$$

both give the x -distance between M and M' .

Similar relations must hold for the y -distance between M and M' .

Note that when M and M' are within the communication radius of each other, they “see” each others states and hence the above comparisons can be made. Initially designated neighbors are within the communication radius of each other and the above described movement rules will never break the connectivity between two designated neighbors.

Then in phase 3, the $right_x$ value of M is replaced by (6.1), and the $left_{x'}$ value of M' is replaced by (6.2). After phase 3 all movement values are made to be zero. Then

the sensor again goes to the movement phase, and determines the next movement direction according to the rules described above.

Note that the description of the state changes is given here for the sake of completeness but in the actual simulator the tracking of the left and right neighbors can be done more straightforwardly.

When we apply the above rules together we find that cycles occur in some cases. For example, at time t , one sensor moves in one direction and at the next time period it moves back to the opposite direction and this pattern of movement continues. One such example is shown in Figure 6.2.

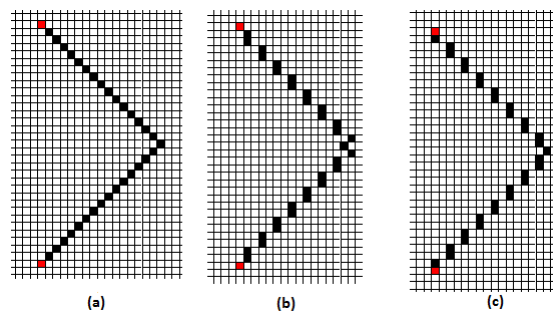


Figure 6.2: An Example of a Cycle

In Figure 6.2 two end points are marked as red. If we are given the initial configuration as 6.2(a), then if we apply the above mentioned rules then after some period the sensors fall in a cycle and move between configuration 6.2(b) and 6.2(c).

The cycle mostly occurs due to the last rule of the above mentioned rules. To overcome this problem, we introduce two different algorithms (probabilistic and deterministic) each having one additional rule added to the above mentioned rules. In the case of these modified algorithms a cycle occurs only at the very end between different shortest paths between the two end points.

Probabilistic Algorithm: The additional rule of this algorithm is : if a sensor satisfies the last rule (one neighbor is on the axis and another is in either positive or negative direction) then the sensor draws a number between 0 and 1 and if the number is less than a predefined threshold (p) then the sensor moves and otherwise does not move. This rule is applied for both the vertical and horizontal axis.

Deterministic Algorithm: The additional rule is: at each time period t , each sensor for each axis, remembers the direction of its last move and after satisfying any rule if it finds that it tries to move in the opposite direction, then the sensor does not move; otherwise it moves. If the sensor does not move at time $t - 1$ then this rule does not apply at time t .

6.4 Simulations and Results

We implement the algorithms in Python version 2.7. We consider two metrics to compare the algorithms with the optimal solution [42].

1. Height (h): The height of a configuration at time t is the maximum distance between a mobile sensor and a shortest line connecting two end points at time t . Note that, $h = 0$ in an optimal solution.
2. Length (l): The length of a configuration at time t is the sum of distances between designated neighbors along the line of sensors from one end point to another. The length is calculated as the distance that needs to be traveled by a mobile sensor starting from one of the end points to the other while visiting all the intermediate sensors. In an optimal solution the length is the number of cells in a shortest path between the two end points.

We consider four different initial chain configurations as in Figures 6.1, 6.3, 6.4, and 6.5 respectively, and compare the results given by the deterministic and probabilistic algorithms.

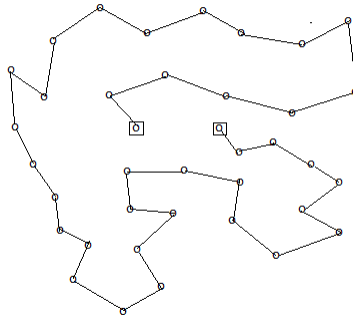


Figure 6.3: Scenario 2: A Chain Configuration

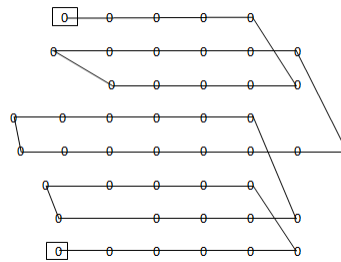


Figure 6.4: Scenario 3: A Chain Configuration

In the case of scenario 1, both algorithms give the optimal solution in 2 time periods. Note that in this case if we consider different numbers of sensors with the same initial height then the time required to give the optimal solution does not change since the time depends on the height. For example, in the case of Figure 6.1, even if we consider more sensors with same initial height ($h = 2$) still we will get the optimal solution in 2 time periods. This is not applicable for different complex chains such as scenarios

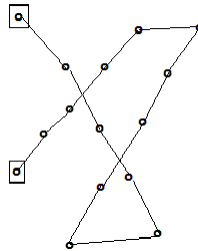


Figure 6.5: Scenario 4: A Chain Configuration

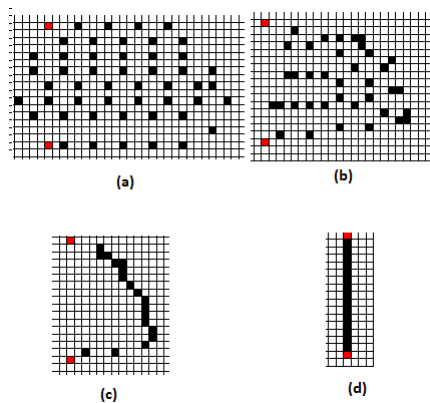


Figure 6.6: An Example of a Simulation: (a) Initial Configuration, (b) Configuration after 5 Time periods, (c) Configuration after 20 Time periods, and (d) Final Configuration

2 and 3.

We consider different numbers of sensors (25, 50, 100, 200) for scenarios 2, 3 and 4 and find that deterministic algorithm gives us the optimal solution in all the cases. The cycle occurs if and only if multiple shortest paths exist in a solution.

We consider different probability values (p) in the case of the probabilistic algorithm and run each experiment ten times for each p . We find for $p > 0.5$, the probabilistic algorithm does not give the optimal solution in some cases. There are some cycles in the solutions that occur quite early in the simulations but in the case of $p \leq 0.5$, we do not find any early cycles often and the algorithm gives us the optimal solutions

most of the time. However, there are some cycles present in these cases too but they exist at the very end of the simulations among the multiple shortest paths between the two end points.

In the case of probabilistic algorithm we find that time taken by the algorithm to give the optimal solution increases with the decrease of p .

We present time taken by the deterministic and probabilistic algorithms for different simulations in Table 6.1. We find that deterministic algorithm is faster than the probabilistic algorithms and in the case of probabilistic algorithms, the algorithm with $p = 0.5$ gives us the solution earlier than any other p , where $p \leq 0.5$. Note that, we do not include the time taken by the probabilistic algorithms with $p > 0.5$ in the table as they sometimes fall in early cycles and in that cases the final solutions are far from the optimal in terms of the length and the height.

It is also noted that there is no relationships between the results of the two different scenarios. The time required to reach the optimal solution totally depends on the initial configurations.

An example of the simulation for the case of scenario 3 is shown in Figure 6.6 which shows the initial, two intermediate (at time, $t = 5$ and $t = 20$), and the final configurations.

6.5 Summary

In this chapter of the thesis we considered the minimizing chain length problem where all the sensors except the two end points of a long windy chain move in such a way to minimize the length of the chain between the two end points. We proposed

| <i>Scenario</i> | <i>Sensors</i> | <i>Deterministic</i> | <i>Probabilistic</i> | | |
|-----------------|----------------|----------------------|----------------------|-----------|-----------|
| | | | $p = 0.5$ | $p = 0.4$ | $p = 0.3$ |
| 2 | 25 | 18 | 50 | 120 | 230 |
| 2 | 50 | 24 | 78 | 165 | 260 |
| 2 | 100 | 31 | 92 | 255 | 310 |
| 2 | 200 | 40 | 105 | 285 | 370 |
| 3 | 25 | 34 | 300 | 550 | 780 |
| 3 | 50 | 41 | 340 | 620 | 830 |
| 3 | 100 | 49 | 400 | 670 | 900 |
| 3 | 200 | 60 | 520 | 720 | 940 |
| 4 | 25 | 20 | 24 | 42 | 68 |
| 4 | 50 | 28 | 37 | 55 | 72 |
| 4 | 100 | 34 | 53 | 60 | 77 |
| 4 | 200 | 43 | 67 | 68 | 86 |

Table 6.1: Comparison of Algorithms for Different Scenarios with Different Numbers of Sensors

deterministic and probabilistic algorithms and found that the deterministic algorithm gave us the optimal solutions in practice faster than the probabilistic algorithms.

Chapter 7

Object Monitoring in MWSN

Object monitoring is a well studied application of mobile wireless sensor networks [120, 107]. In a general case, a number of objects need to be monitored by the sensors and the objects can be static or mobile. Monitoring mobile objects is considerably harder than monitoring static objects. Mobile sensor networks can be useful in monitoring animal behavior. In this case, animals can move randomly within a network and the mobile sensors can also move to monitor their behavior and other aspects. Note that we are considering the applications where the sensors do not send the data in the real time. They can store the data and send it to the base stations later or some mobile sinks can be used to gather the data from the sensors.

Algorithms for different variants of the object monitoring problem can be found in [103, 111]. The solutions provided by these algorithms are either centralized or distributed. Islam *et.al.* [63] propose a local algorithm for a variant of the object monitoring problem where the objects are stationary.

In this section, we design a cellular automaton based local algorithm for an object

monitoring problem. We consider mobile objects and to the best of our knowledge this is the first cellular automaton based algorithm that has considered this application of mobile sensor networks.

This chapter is organized as follows: The problem being addressed, namely, modeling a mobile wireless sensor network as a cellular automaton for the object monitoring problem, and the algorithm for solving it, are presented in section 7.1 We discuss simulations and results in section 7.2. Finally, we summarize the chapter in section 7.3.

7.1 Problem Definition and Algorithms

A set of mobile sensors (m_1, m_2, \dots, m_n) are deployed in a two dimensional sensor network. A number of mobile objects (o_1, o_2, \dots, o_k) are also deployed in the network. Typically we consider situations where $k < n$. The objects can move randomly within the network which has a bounded area and the mobile sensors try to move accordingly to monitor them as long as possible. We consider the movement speed of the mobile objects to be higher than the mobile sensors, because otherwise, the problem becomes trivial. We consider that the mobile sensors can move one cell in one time unit from their current cell using the algorithm. They can move diagonally too whereas the mobile objects choose any location within a radius-2 neighborhood in random and can move there. So the speed of the mobile objects can be twice compared to the mobile sensors.

Initially all the sensors and objects are deployed densely within the network where each object is monitored by at least one of the sensors. However, as soon as the objects start moving they are not guaranteed to be monitored if the sensors do not

move accordingly. The sensors use the following CA based movement algorithm in order to maximize the number of objects monitored and the length of the time they are monitored.

Neighborhood: A sensor monitors objects that are within its sensing radius. A sensor communicates with other sensors within its communication radius and uses the information obtained to determine the movement direction.

State Representation: The state of a cell gives the number of mobile sensors located in that cell and the locations and positions of the objects monitored by sensors in the cell. Thus, when the sensing radius is one, the state of a cell C is a 10-tuple

$$(l, o_1, \dots, o_9),$$

where l is the number of sensors in cell C and o_i , $i = 1, \dots, 9$ give the numbers of objects in each of the cells with radius one from cell C .

Note that if the state would simply store the number of objects within sensing radius from C , when a sensor m communicates with two different cells C_1 and C_2 with overlapping sensing radius, the sensor m would have no way of knowing how many of the objects within the sensing radii of C_1 and C_2 , respectively, are “in common”. For this reason the state of a cell needs to store the precise location of each object within its sensing radius.

Please note that the actual implementation in the simulator is simpler, the above mentioned states would be needed if these are implemented in “real CA”.

Movement Rules: The movement rule for each of the sensors is described below for the X axis. Similar rules are applicable for the Y axis. This algorithm has two phases. In the first phase, a sensor computes its future location and in the second

phase it decides whether it should move to that location or not.

First Phase:

- For the X axis, each sensor counts the number of mobile sensors in the positive and negative directions within R_c radius. It also counts the number of objects which is the sum of the number of objects the sensor finds in its sensing radius and the number of mobile objects its neighbors (those that are within its communication range) find in their respective sensing radius. Note that an object can be sensed by multiple sensors but it is counted only once.
- If the sensor does not find any object in one of these two directions then it moves towards that direction that has at least one object.
- If both directions have no object then the sensor applies the following rules. These rules are similar to the rules considered in [29, 31].
 - It moves towards that direction that has fewer sensors.
 - If there is no mobile sensor in both directions or the directions have the same number of sensors then it does not move.

In this case, even though a sensor does not see any object in its communication radius, it wants to move in the hope that in the future it can detect and monitor the objects. Moreover, it does not need to stay in the current position as there are other sensors already there to monitor the shared region.

- If a sensor finds that both directions have at least one object then it calculates the following ratio for the positive X direction.

$$ratio_{xpos} = \frac{n_{xpos}}{k_{xpos}} \quad (7.1)$$

where n_{xpos} is the total number of neighboring sensors, the sensor sees within distance R_c in the positive x -direction and k_{xpos} is the total number of objects sensed by the sensor and its neighbors in the positive x -direction.

- It calculates $ratio_{xneg}$ similarly for the negative X direction.
- Now the sensor decides to move toward that direction that has the smaller ratio. The idea is to move towards that direction where the objects have smaller number of sensors on average to monitor them. If the ratios are equal then the sensor does not move.

Second Phase: Now the sensor moves to the location it has already chosen above if the location is empty. If the location is not empty then the sensor does not need to go there as there is already at least one sensor to monitor the objects of its R_s neighborhood. However, this rule does not totally eliminate the chance of having multiple sensors in a cell.

If any cell has more than one sensor at a given time then only one sensor applies the above mentioned algorithm and moves if it satisfies the rules. If the sensor does not satisfy any rule that allows it to move, then the sensor randomly chooses any location from its radius 1 neighborhood and moves to that location. Other sensors do not move at that time period.

Now we consider one example that illustrates the rules. Figure 7.1 shows a two dimensional mobile sensor network with $R_c = 2$. There are three mobile sensors

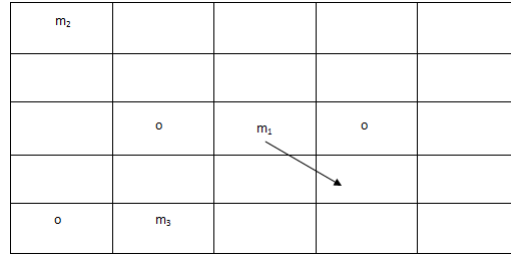


Figure 7.1: An Example of of Object Monitoring Problem with $R_c = 2$ and $R_s = 1$

in this network, m_1, m_2 , and m_3 . The objects are denoted as o . Now we describe the next location of mobile sensor m_1 . In order to determine the movement in the X -direction, the sensor finds objects both to the left and to the right of its current position. Thus, the algorithm calculates $ratio_{xpos}$ (right side) and $ratio_{xneg}$ (left side) and finds that the right side has the minimum ratio. Consequently, it chooses the right side as the next movement direction.

To determine the movement in the Y -direction, the sensor finds no object above of its current position. However, it finds one object below. So it chooses to go in the negative direction as the next movement direction.

Since the sensor decides to move diagonally right and down and the location is empty then the sensor moves towards that location.

7.2 Simulation and Results

Our algorithm is implemented in Python Version 2.7. The simulations use different grid sizes, different numbers of mobile sensors, and different numbers of mobile objects. We consider $R_c = 3$ or $R_c = 2$. In all the cases, the sensing radius $R_s = 1$.

Initially we deploy different numbers of mobile sensors and the objects in a square

block as in [29, 31] and chapter 4 of this thesis. For example, for 200 mobile sensors and 100 mobile objects we choose a 15×15 block in the middle of a 100×100 grid. For each sensor and object, we randomly pick a location from this block where it is deployed. Note that, at any given time, a cell can have more than one sensor or more than one object. We also specify that in the initial deployment each cell can not have more than three sensors and each cell can have maximum five objects. However, there is no such bound after the deployment. These bounds are only for the simulation purposes. The algorithm is general and can use different such bounds. An object is considered to be monitored if it is in the sensing radius of at least one mobile sensor. Initially, typically, all the mobile objects (100%) are monitored by the mobile sensors. However, when the objects start moving this percentage gets lower but after a certain period of time (time depends on the simulation parameters, for example, grid size and the number of mobile sensors) this number does not change much. Please note that, designing a local algorithm that provides 100% monitoring is quite difficult to achieve, and when the grid size is large compared to the number of sensors achieving monitoring of all the objects is obviously impossible. We are interested to get a competitive percentage that can be applicable in different real life applications.

We also compare our monitoring algorithm with the dispersion algorithms described in [30, 29] (chapter 4). These algorithms are designed only to distribute the mobile sensors to cover a large area of the network while our current algorithm determines the movement of the sensors also based on information about nearby objects to be monitored. However, when we compare the algorithms we find that the algorithm proposed in [30] performs considerably worse compared to the other algorithms. For

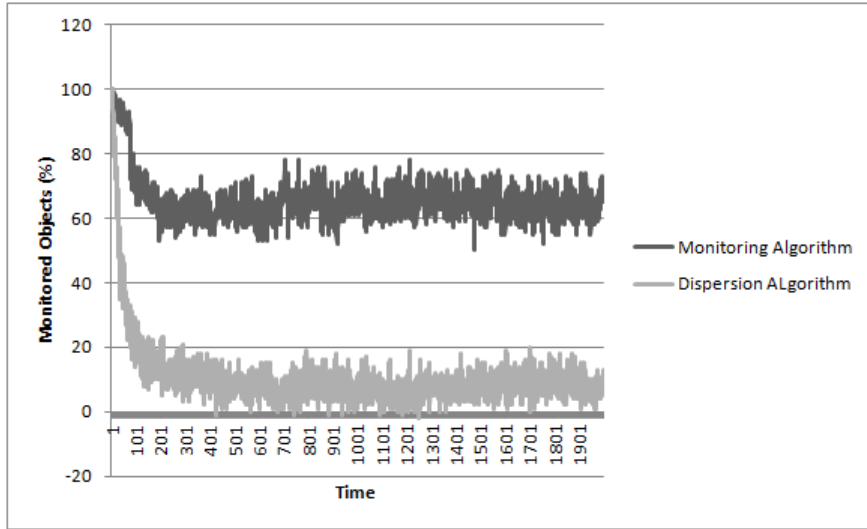


Figure 7.2: Comparison Results for 200 Mobile Sensors and 100 Mobile Objects with $R_c = 3$ and with 100×100 Grid

this reason, we do not present the results for this algorithm in the rest of this section. In the next part of this section, we call the algorithm of [29] as *the dispersion algorithm* while the algorithm proposed in this section is called *the monitoring algorithm*. As connectivity is not an issue in this application, we do not consider the blocking and the move-back rules of the dispersion algorithm proposed in [29].

Figure 7.2 shows a comparison of the results of the monitoring and the dispersion algorithm where we consider a grid of size 100. We also consider 200 mobile sensors with $R_c = 3$ and 100 mobile objects. We find that the monitoring algorithm performs better (percentage never goes below $\approx 45 - 50$) than the dispersion algorithm which does not explicitly consider the presence of the mobile objects. We run each experiment 20 times and show the average results.

We also find that we can get a much improved result if we increase the number of mobile sensors without changing the number of mobile objects. For example, if we

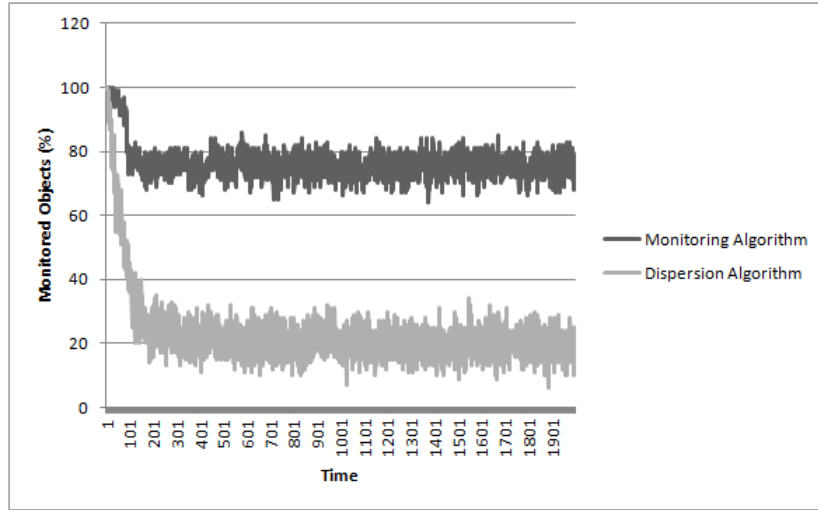


Figure 7.3: Comparison Results for 300 Mobile Sensors and 100 Mobile Objects with $R_c = 3$ and with 100×100 Grid

increase the number of mobile sensors to 300 without changing any other parameters of the previous simulation then we get much improved solutions for both the algorithms. This is quite expected as the network is bounded and increasing the number of mobile sensors increases the total coverage of the network and eventually more objects can be monitored. Figure 7.3 shows the comparison for this case.

We present different simulation results for different parameters in Table 7.1 and Table 7.2. The first column of the table represents the total grid size. We consider three different grid sizes (100, 125, 150) and in each case we consider 100 mobile objects and different numbers of mobile sensors with $R_c = 3$ (Table 7.1) and $R_c = 2$ (Table 7.2). Note that in all the experiments we consider $R_s = 1$. We present the results of the monitoring and the dispersion algorithms in columns 3 and 4, respectively.

The percentage value refers to the fraction of objects that is monitored after running the algorithm for a sufficiently long time so that the value, more or less, stabilizes. Naturally there remain some fluctuations in the precise number of objects monitored

due to the random movement of the objects. With the numbers of sensors considered in the experiments of Figures 7.2 and 7.3 the “stabilized state” is reached after roughly 200 – 300 time steps.

Note that, we report the minimum percentages (the lower bound) for both the cases. For example, in case of Figure 7.2 we report 50 and 10 (from Table 7.1) respectively. In all these cases, the monitoring algorithm outperforms the dispersion algorithm. We run similar experiments for different numbers of mobile objects (50, 150 , and 200). As the results are similar (monitoring algorithm outperforms dispersion algorithm) we do not present the results explicitly.

| Grid Size | Number of Sensors | Monitoring Algorithm (%) | Dispersion Algorithm (%) |
|-----------|-------------------|--------------------------|--------------------------|
| 100 × 100 | 200 | ≈ 50 | ≈ 10 |
| 100 × 100 | 300 | ≈ 70 | ≈ 12 |
| 125 × 125 | 200 | ≈ 50 | ≈ 8 |
| 125 × 125 | 300 | ≈ 68 | ≈ 10 |
| 125 × 125 | 400 | ≈ 75 | ≈ 25 |
| 150 × 150 | 200 | ≈ 45 | ≈ 7 |
| 150 × 150 | 300 | ≈ 55 | ≈ 10 |
| 150 × 150 | 400 | ≈ 65 | ≈ 15 |
| 150 × 150 | 500 | ≈ 75 | ≈ 20 |

Table 7.1: Experimental Results on Object Monitoring Problem with 100 mobile objects and $R_c = 3$

We also analyze the confidence interval of the mean of the results of these two algorithms. We use the 95% confidence level to calculate the values. So the confidence interval of the mean is $\text{mean} \pm 1.96 \times \frac{\text{Std}}{\sqrt{N}}$ [37], where *Std* is the standard deviation of the population and N is the size of the population ($N = 20$ in our simulations). Using this formula, we find that the confidence interval of the mean of our experiments is

| Grid Si | Number of Sensors | Monitoring Algorithm | Dispersion Algorithm |
|------------|----------------------|-------------------------|-------------------------|
| 100 × 100 | 200 | ≈ 45 | ≈ 8 |
| 100 × 100 | 300 | ≈ 65 | ≈ 9 |
| 125 × 125 | 200 | ≈ 42 | ≈ 7 |
| 125 × 125 | 300 | ≈ 56 | ≈ 8 |
| 125 × 125 | 400 | ≈ 67 | ≈ 10 |
| 150 × 150 | 200 | ≈ 40 | ≈ 6 |
| 150 × 150 | 300 | ≈ 50 | ≈ 8 |
| 150 × 150 | 400 | ≈ 60 | ≈ 8 |
| 150 × 150 | 500 | ≈ 68 | ≈ 12 |

Table 7.2: Experimental Results on Object Monitoring Problem with 100 mobile objects and $R_c = 2$

quite narrow compared to the average values. This is because the standard deviations of the values are very small. As the confidence interval is very narrow we do not show it in the figure.

7.3 Summary

We proposed a cellular automaton based algorithm for the mobile object monitoring problem in a mobile wireless sensor network where both the objects and the sensors can move and our main goal is to monitor the objects as long as possible. To the best of our knowledge, this is the first cellular automaton based local algorithm proposed for this problem. The simulation results indicated that our proposed algorithm worked well in practice and could monitor a good number of mobile objects constantly.

Chapter 8

Underwater Mobile Sensor Networks

8.1 Introduction

Underwater wireless sensor networks have attracted considerable attention due to their wide applications [99, 55]. A typical UWSN consists of a number of underwater sensors and a limited number of surface relay stations. The underwater sensors use acoustic signals to communicate with each other and with the relay stations. On the other hand relay stations use radio communication with on-shore stations. Relay stations normally stay on the surface while sensors can go deep in the water and can collect different types of data. These data are routed to the on-shore stations through multi hop communication between the sensors and the surface relay stations. Figure 8.1 illustrates the general set-up.

Two of the most important metrics to evaluate a UWSN are the coverage and the

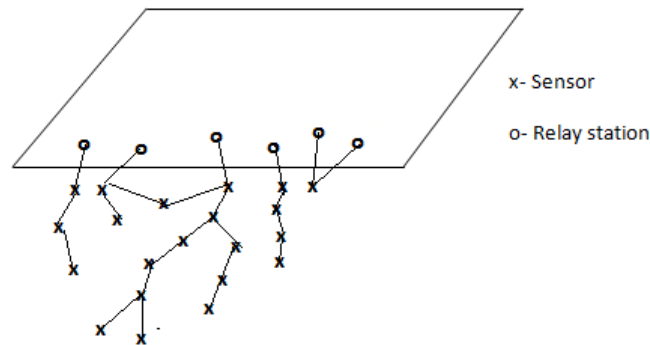


Figure 8.1: An Example of Underwater Networks

connectivity. Coverage is defined as the area that is monitored by the sensors while connectivity ensures that each sensor is connected to at least one of the surface stations so that data can be routed from each sensor to the surface stations. It is hard to implement a global algorithm in a UWSN which can maximize the coverage while maintaining the connectivity. For this reason, initially sensors are deployed close to the relay stations and they are allowed to disperse vertically by themselves to improve the coverage while maintaining the connectivity.

Different variants of the mobile dispersion problem have been considered for the two dimensional case [71, 68, 88, 58, 111, 36]. Most of the algorithms for these problems are distributed. A few CA based algorithms have also been proposed [29, 31, 106] but to the best of our knowledge there is no fully local algorithm available for three dimensional or underwater sensor networks. All the proposed algorithms for UWSN are distributed [3, 44, 45]. In all the cases, they consider a fixed area and try to maximize the coverage of that area. They also have prior information about the depth of the water. In our case, we consider an unbounded area and the sensors do not have any prior information about the depth of the water. However, we can also implement our algorithm for a fixed depth of water.

We explain one deterministic algorithm that gives us optimal solutions for different initial deployments (explained in section 8.3). However, the final positions of the sensor consists of a chain which is not a reliable solution.

We propose two other algorithms. Our experiments have shown that using synchronous deterministic CA rules in these algorithms, among other problems, often leads to infinite cycles. In order to improve the algorithms performance the rules are applied probabilistically.

At each time step, a sensor with probability p checks the states of its neighborhood cells and determines whether or not to move. Each sensor draws a number between 0 and 1 and if the number is less than a threshold (p), where $0 < p < 1$, then the sensor checks the rules and decides whether it should move or not. The value of p is a parameter and can be selected in a way to improve the performance of the algorithm. Our algorithms use very limited local information and give very competitive results in terms of coverage while maintaining connectivity.

In section 8.2 we define the depth adjustment problem of a UWSN and the system model of the network. Our algorithm is described in section 8.3. Simulation results of this algorithm are explained in section 8.4. Finally we summarize the chapter in 8.5.

8.2 Problem Definition and System Model

The *Depth Adjustment Problem* of a UWSN is defined as: A total n mobile sensors are initially deployed near to the surface level so that all the sensors are connected to at least one of the fixed relay stations located at the surface. The sensors can move

only in one dimension; up or down in the water. We refer to the vertical axis as the Z -axis, while X and Y refer to the horizontal plane. Our main goal is to disperse the sensors in a way that maximizes the coverage while maintaining connectivity with relay stations at the surface level.

Each sensor can cover the region of the cells that are within its sensing radius (R_s) and communicate with any sensor that is within its communication radius (R_c). If two sensors are close together then they are connected with each other but on the other hand they share some coverage. So the main trade-off is between coverage and connectivity. We try to increase coverage by reducing the sharing region without losing the connectivity.

We study the depth adjustment problem experimentally for cases where (R_c, R_s) is $(3, 1)$, $(3, 2)$ or $(2, 1)$. It is easy to see that when $R_s \geq \frac{1}{2}R_c$, the sensor deployments that maximize total coverage of the network consist of long chains of sensors that would not be useful in practice, for example, because they become easily disconnected. For this reason we measure the performance of the algorithms in terms of strongly connected coverage described in section 4.5 of this thesis.

8.3 Algorithms

We consider a three dimensional cellular automaton model for this depth adjustment problem. For example, if we consider a radius 1 Moore Neighborhood in the three dimensional case then each cell has 26 neighbors. So if we consider $R_s = 1$, and if we place a sensor in a cell of 3 dimensional grid then each sensor can cover 27 cells (including itself).

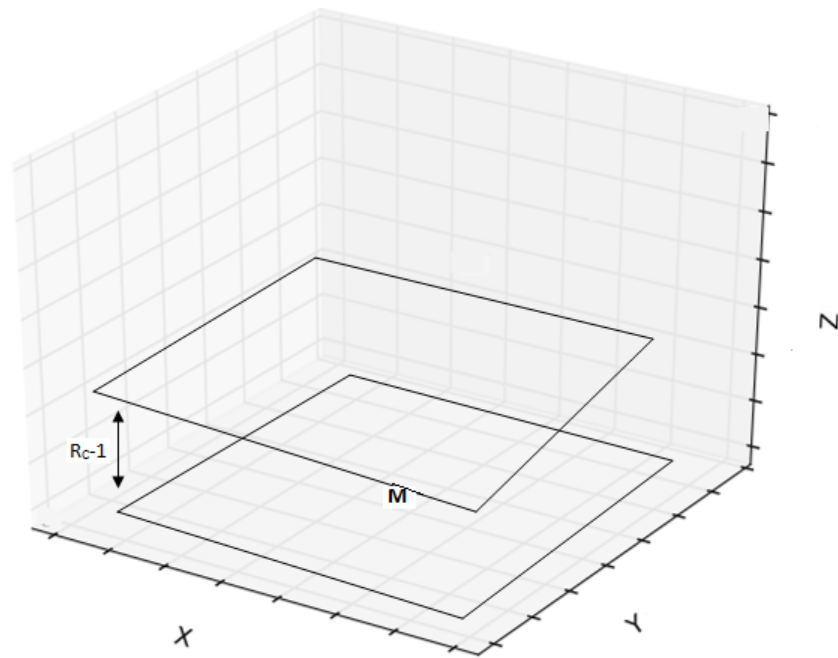
Next we describe the CA rules that control the movement of the sensors in our algorithm:

The state of each cell indicates the number of sensors in that particular cell. If there is no sensor then the state of the cell is 0. Each sensor can only move upwards or downwards along the depth of the grid. At any period of time, each sensor can be in one of the three states. If the sensor does not move at period t then at $t + 1$, the state will be 0. If the sensor moves downwards or upwards then the state will be 1 or -1 , respectively. The purpose of remembering the last movement direction will be explained below when defining the move-back rules.

The general idea of the movement rules is that each sensor tries to move down (along the Z axis) to increase the coverage. However, if this sensor is connected only with sensors that are R_c distance back along the Z axis, then moving down can break the connectivity (if the other sensor does not move down too). For this reason we introduce a *blocking rule*. The purpose of this rule is to maintain the connectivity.

Figure 8.2 shows the blocking rule for a sensor M . Before moving down along the Z axis it checks whether there is any neighbor within distance $R_c - 1$ away along the Z axis. If it finds at least one other sensor within the rectangle of size $R_c \times (2R_c + 1) \times (2R_c + 1)$ depicted in Figure 8.2, then it moves down and changes its state to be 1, otherwise, the sensor does not move and the state will be 0.

This blocking rule is not sufficient to always maintain connectivity. For example, at any given time, if two neighboring sensors are within the same depth level then when they apply the blocking rule they can see each other and both can move. However, the closest sensor to them in the positive Z direction may be at distance R_c of their

Figure 8.2: Blocking Rule with $R_c = 3$

original position, which means that by moving down they lose the connection to it. So this sensor will be disconnected after that time period.

One way to fix the breaking of connectivity is to make the “blocking area” asymmetric in a way that two sensors cannot see each other within their respective blocking areas. The blocking rule is modified so that on the same horizontal plane a sensor M looks only at exactly half the plane. The half plane is illustrated in Figure 8.3 with the case $R_c = 3$.

Strictly upwards from M , the sensor looks at the $R_c - 1$ rectangles of size $2R_c + 1$ as before. The modified blocking rule guarantees that connectivity is always preserved and the algorithm produces optimal coverage in the case $R_c = 3$, $R_s = 1$ and good coverage also in the other cases. Note that for this algorithm the rules could even be applied deterministically and synchronously. However, the problem is that the final

We apply our algorithm for three different (R_c, R_s) pairs. In the case $R_c = 3$ and $R_s = 1$, an algorithm based on movement rules as defined above obtains optimal solutions for various values of the probability parameter p . In this case, the optimal solution has coverage value $n \times 27$ and all the sensors are connected to some relay stations at the surface level. The details of the simulation results are presented in the next section. However, when we apply the algorithm for the $(3, 2)$ and $(2, 1)$ cases, the results are further from the optimal because of always overlapping sensing region among nearby sensors. That is why we consider following a *quadrant rule* that makes sensor movement more aggressive but still maintains connectivity.

In order to disperse the sensors more aggressively, we use a weaker blocking rule that, roughly speaking, looks at sensors at distance up to R_c in the positive Z direction as long as there are other sensors in the same quadrant that are on the same horizontal level as the current sensor.

For the sake of readability, we explain the extended blocking rule in a two dimensional model in Figure 8.4, where Z denotes the vertical axis and X is the only horizontal axis. Afterwards we explain how the model is extended for three dimensions.

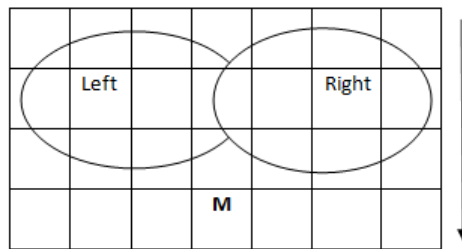


Figure 8.4: Extended Blocking Rule at 2D with $R_c = 3$

The left quadrant of the sensor M consists of $(R_c + 1) \times R_c$ cells upwards and towards the left of M that M sees in the positive Z direction. The right quadrant is defined

similarly.

The extended blocking (quadrant) rule is defined as follows: M can move down if it sees another sensor in the positive Z direction within distance $R_c - 1$, or if it sees a sensor in the left quadrant and another sensor within distance R_c horizontally to the left of M , or if it sees a sensor in the right quadrant and another sensor within distance R_c horizontally to the right of M .

The idea of the extended blocking rule is that, although M may lose direct communication to the sensor(s) in the positive Z direction, the existence of the other sensor at the same horizontal level as M in the same quadrant will preserve the connection, unless these sensors decide to move down at the same time.

For the three dimensional case, the idea is exactly the same. Now we have four quadrants that are called North East (NE), North West (NW), South East (SE), South West (SW). In this case, each quadrant consists of $(R_c + 1) \times (R_c + 1) \times R_c$ cells

If a cell has more than one sensor, a sensor is randomly selected which checks the blocking rules and if it satisfies the blocking rule then it move downwards otherwise it stays at that location. All other sensors can move downwards without even checking the blocking rules.

8.4 Simulations and Results

We have implemented our algorithms in Python Version 2.7. We consider two different deployment schemes.

First, we pick a surface area and deploy the same number of sensors at all surface

points. For example, in the case of a total of 75 sensors, we pick 15 points and deploy 5 sensors in each of these points. However, our algorithm can also handle a different numbers of sensors for a different number of points.

For the second type of deployment, initially we deterministically pick (x, y) locations. Figure 8.5 depicts the locations of the surface cells where sensors are deployed as viewed from above. Each of these cells is at a distance R_c from its neighbors. Then we deploy the same number of sensors in different depths of each point.

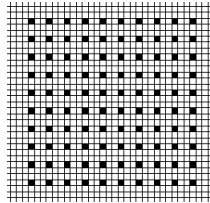


Figure 8.5: Second Type of Deployment with $R_c = 3$

We run our algorithm 10 times for probability values, $p = 0.1, 0.2, 0.3$ and 0.4 with different initial deployments. In case of $(3, 1)$ we get the optimal solution for all the cases and 0.4 gives the solution in the least amount of time. The final deployment resulting from one such simulation is shown in Figure 8.6. In this example, initially we deploy 75 sensors in a total of 25 locations of the surface. Each location has a total of 3 sensors. When we apply our algorithm with $p = 0.4$ it gives us the optimal solution every time (10) for both deployment schemes in around 250 time periods on average. Sensors are marked as “x” in the figure and the relay stations at the surface are marked as “o”. The final positions of the same numbers of sensors for the second type of deployment scheme is shown in Figure 8.7.

Figure 8.8 shows a final configuration for the first type of deployment scheme where a total of 300 sensors are deployed in a total of 25 points each having a total of

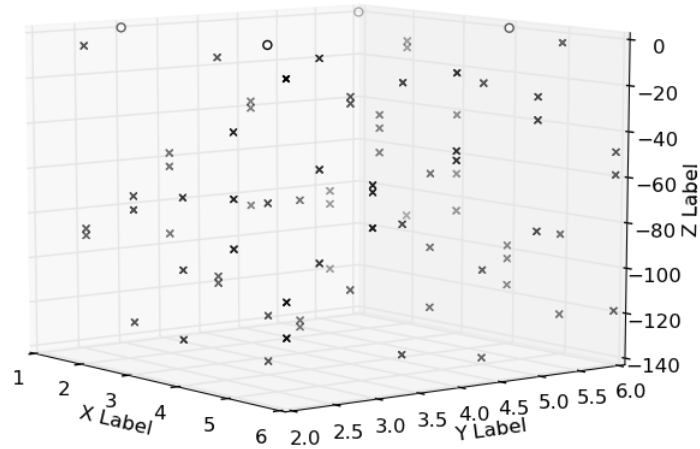


Figure 8.6: Final Positions of Total 75 Sensors with $R_c = 3$ and $R_s = 1$ When Sensors Initially the Sensors are Deployed at all Surface Points.

12 sensors. We also plot the increase of the coverage of the network with time in Figure 8.9 where we see that for $p = 0.4$ we get the optimal solution faster than in the other cases. In every case there are some sudden falls of coverage and this is higher in the case of $p = 0.4$. This is due to the break of connectivity explained in the previous section. However, in every case, the algorithms restore the coverage because of the move back rules. In the case of $p = 0.4$ it restores the coverage quickly as the sensor checks back with higher probability compared to the other cases where as in the case of $p = 0.1$ it takes some time to restore the coverage.

As discussed in section 8.2, for the cases where (R_c, R_s) is $(3, 2)$ or $(2, 1)$, instead of measuring just the coverage we should measure the strongly connected coverage. We compare the results to the benchmark configuration depicted in Figure 8.10. Roughly speaking, the benchmark configuration is obtained as follows. We first place sensors in a rectangular grid so that the nearest neighbor of each sensor in the positive and negative X, Y, Z directions is exactly at distance R_c . From this placement we then remove every other sensor, from adjacent columns we always remove sensors at

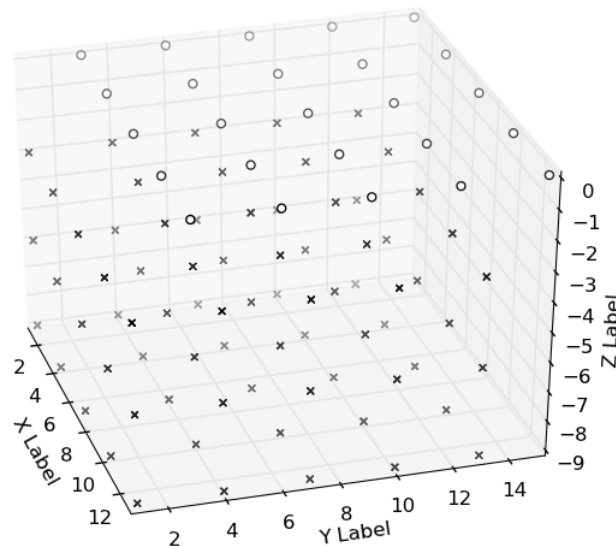


Figure 8.7: Final Positions of Total 75 Sensors with $R_c = 3$ and $R_s = 1$ at Some Surface cells. Each Cells are at Distance 3 from the Neighboring Cell.

alternating levels.

We use the algorithm employing the extended blocking rules. Figure 8.11 depicts the final positions of sensors in one run of the algorithm in case where the number of sensors is 75 and $R_c = 3$. Note that the final solutions returned by the algorithms vary only with the change of R_c . Figure 8.11 is also an optimal solution for $R_s = 1$ but not for $R_s = 2$. In the latter case we compare the solutions with the benchmark configuration (Figure 8.10) in terms of average strongly connected coverage. We also consider a similar comparison for the (2, 1) case.

The comparison results for different numbers of sensors in two different deployment schemes are shown in Figures 8.12 and 8.13. The Y axis represents the ratio (raSCC) of the average strongly connected connectivity (in percentage) for the results produced by the algorithm and for the benchmark configuration. We find from this comparison that our algorithm gives competitive results compare to the benchmarks.

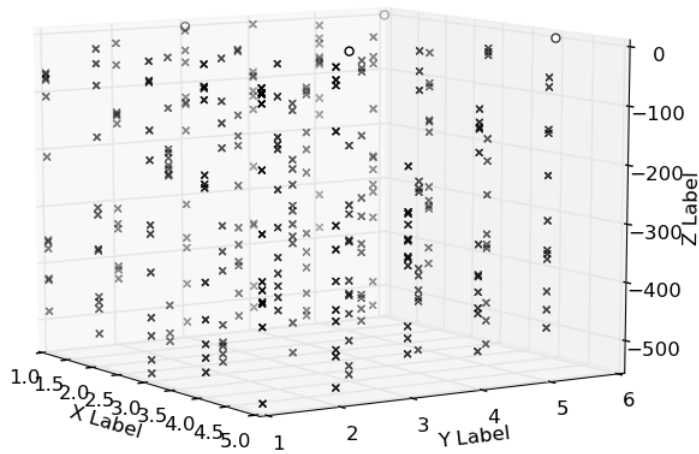


Figure 8.8: Final Positions of the Sensors (300) for $R_c = 3$

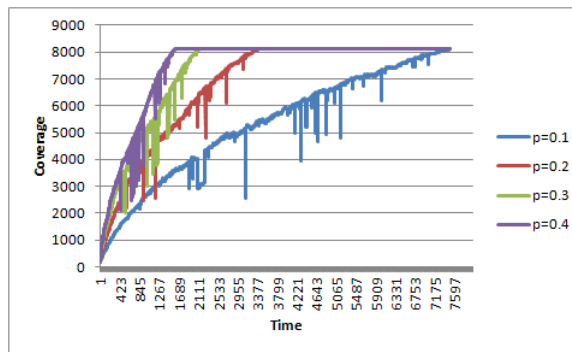


Figure 8.9: Coverage with Time for 300 Sensors with $R_c = 3$

In all of the above mentioned simulations, we find that all the sensors are connected to the surface stations in the final configurations. All the simulations terminated without cycles and in a very reasonable time. For example, in case of 300 sensors, it takes on average 1500, 1800, 3000 and 7500 time periods for four different p values, 0.4, 0.3, 0.2, 0.1, respectively.

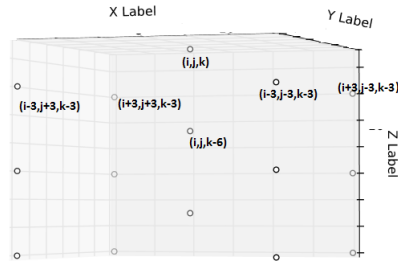


Figure 8.10: Benchmark Configuration with $R_c = 3$.

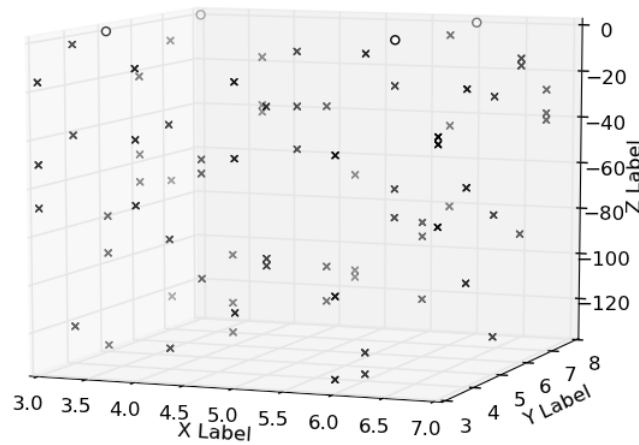


Figure 8.11: Final Positions of the Sensors with Extended Blocking Rules for $R_c = 3$

8.5 Summary

We proposed cellular automaton based probabilistic algorithms for the depth adjustment problem in underwater sensor networks where sensors are allowed to move only vertically. Our algorithms used very limited information to compute the possible future locations of a sensor. We considered different (R_c, R_s) pairs and found that in the case of $(3, 1)$ we got the optimal solution even with very simple rules. In the other two cases we used extended blocking rules to get competitive results compared to a benchmark.

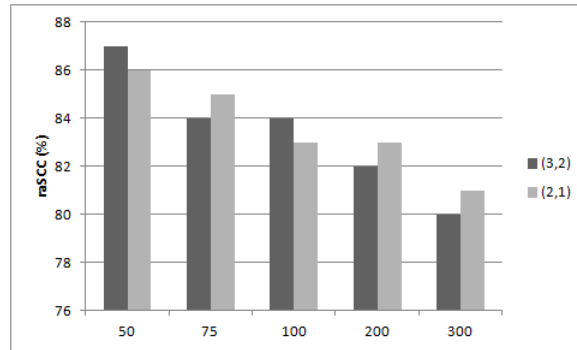


Figure 8.12: Strongly Connected Coverage Produced by the Algorithm (as a Percentage of the Benchmark) When Initially the Sensors are Deployed at All Surface Points.

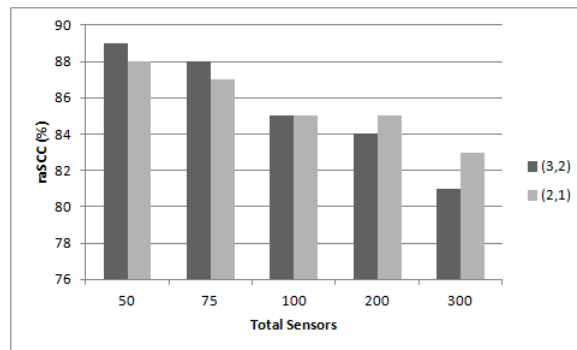


Figure 8.13: Strongly Connected Coverage Produced by the Algorithm (as a Percentage of the Benchmark) When Initially the Sensors are Deployed at Some Surface Cells. Each Cells are at Distance R_c from the Neighboring Cells

Chapter 9

Summary and Conclusions

9.1 Summary

Wireless sensor network is a type of networking technology that differs from the traditional networks in many aspects. Unlike a traditional network, a sensor network has limited power supply, limited memory and processing power. Coverage and the network lifetime are two important metrics considered to evaluate the performance of a sensor network. Since energy is a critical issue, extending network lifetime and maintaining a competitive coverage is a challenge. We proposed a cellular automaton based algorithm for the sleep-wake problem of the wireless sensor networks in chapter 3. Our algorithm gave more network lifetime while maintaining a good amount of coverage compared to an earlier cellular automaton based algorithm. We also used our algorithm for object detection in wireless sensor networks and we found that our algorithm detected more objects than an algorithm known earlier.

It is clear that if we add mobility to the sensors, we can also improve the coverage.

However, adding mobility raises some research issues too. For example, how a sensor decides its movement so that it does not break the connectivity of the network and also improves the coverage. We proposed cellular automaton based algorithms for the two variants of this research issue in chapter 4. In the first variant, we consider a bounded area and sensors are deployed in a dense area of the network and they use CA based algorithms to disperse so that the total area covered by the network is maximized. We put an upper bound on the number of movements for each sensor. We compared our algorithm with an earlier cellular automaton based algorithm for the similar variant and found that our algorithm gave better results and use less time. In the second variant, we considered an unbounded network area and another constraint was added. The sensors need to disperse without breaking the connectivity. We proposed a first set of CA based algorithms for this variant of the problem. We found that for some cases, the optimal solution of this problem consisted of a long chain of sensors which were suboptimal from the fault tolerance point of view. So instead of comparing the solutions of our algorithm with an optimal one, we considered two other benchmarks (square and star). The benchmark configuration were selected to provide good coverage while at the same time each pair of sensors were connected by multiple paths between them. We also designed a new metric which we called strongly connected coverage to compare the solutions. The new metric measured both the area covered by the network and the length of the shortest paths connecting different pairs of sensors. We found that our algorithms performed better than the square benchmark and performed very close to the star benchmark.

Once the sensors are dispersed they need to gather in different geometric shapes for different purposes. We consider the point gathering problem of wireless sensor

networks in chapter 5. In a point gathering problem sensors need to gather in a point without breaking the connectivity of the network. We designed the first CA based algorithm for this problem and found that our algorithm performed better in practice compared to a similar local algorithm.

In section 6 we considered a problem called the minimizing chain length problem where sensors of a windy chain need to be gathered in a straight line between the two end points of the chain. In this case all the sensors except the two end points can move. We designed and implemented two CA based algorithms (one probabilistic and one deterministic) for this problem and found that the deterministic algorithm performed better in practice. To the best of our knowledge, these are the first set of CA based algorithms for this problem.

Object monitoring is an important application of mobile wireless sensor networks. Designing a local algorithm for mobile object monitoring is an important research area. We considered the first CA based algorithm for this problem in chapter 7. We found that our algorithm performed well in practice.

Underwater Sensor networks are also an emergent application of wireless sensor networks. In a typical underwater sensor network, initially sensors can be deployed at the surface of the water and then they can move autonomously to the depth of the water to improve the coverage of the network. However, they need to maintain their connectivity with the surface stations while they move. We considered this problem in chapter 8 and gave a first CA based algorithm for this problem. We compared our algorithms with a benchmark in terms of strongly connected coverage and found that our algorithm performed close to the benchmark.

There are lots of algorithms proposed for these types of problems of wireless sensor

networks. However, most of the algorithms are either centralized or distributed. But a centralized algorithm is not a good solution for these types of optimization problems because of the nature of the applications. On the other hand, distributed algorithms need more message passing which can be a burden for the tiny sensors. Local algorithms are the ideal candidate for these solutions because they need only local information which is very little compared to the whole global information and processing power which can be easily adopted in a sensor network. There are very few local algorithms proposed for different optimization problems of sensor networks. However, most of them are quite complex and hard to implement.

9.2 Future Work

In terms of the application point of view, in future work, we plan to consider hexagonal shapes of neighborhood in our algorithms in future. These are more natural in networking applications. Note that on a square grid when we allow diagonal movement steps the distance between opposite corners of a rectangle is the same as the distance between two lower corners. On the other hand, if we do not allow diagonal movement steps the distance between diagonal corners will be two times the distance between two lower corners of the rectangle. A related issue is that on a square grid two sensors with sensing radius one and that are located at a distance two of each other may jointly cover 1, 2 or 3 cells depending on how they are positioned. On a hexagonal grid they would always jointly cover 1 or 2 cells.

A hexagonal grid would provide a more realistic description of two dimensional topology where distances between different pairs of points would be less distorted. Hence a 2 dimensional cellular automaton model based on a hexagonal grid would be better

suitable for applications, but this kind of models have not been used, probably, because the representation of points on the grid becomes less straightforward.

We also plan to consider Graphics Processing Units (GPU) parallel programming to implement CA based algorithms. They have been proved as a memory efficient and a faster technique.

In the following we list several more specific open questions related to the algorithmic problems studied in the thesis.

- Designing an improved cellular automaton based algorithm for the sleep awake scheduling that does not only count the neighbors but also considers the directions of the neighbors to determine its own state is an open research question. In the current solutions, one sensor changes its state if it finds a specified number of neighbors. However, these neighbors might be in the same direction whereas other directions might not be considered in the corresponding counting. We are interested to investigate this aspect and interested to design such a local algorithm that also takes this issue into the account.
- The probabilistic “forced” sleep periods considered in sleep-wake scheduling in chapter 3 can be viewed as an ad hoc feature that has been added to the cellular automaton model. It remains a question for further research whether it is possible to remove the dips from the coverage in a cleaner way that relies only on CA rules possibly defined on a larger set of states.
- Design an algorithm that will minimize the average hop distance of the problem considered in chapter 4 is an important research challenge. Designing a local algorithm for the mobile sensors that will monitor the mobile objects while not

breaking the connectivity among them is naturally a hard problem. We plan to consider these issues in the future.

- Providing an upper bound of the time taken by any sensor to reach a single position (discussed in chapter 6) in this algorithm is an open research question. We plan to consider this issue along with other gathering problems, for example, gathering in a ring or in a rectangle.
- We plan to extend our research work on various problems in underwater networks. Though we have considered one optimization problem in this thesis, there are lots of other problems that haven't been extensively considered, for example, tracking objects in a underwater network, sink mobility in underwater networks, etc. I strongly believe designing some local algorithms for these problems will help different applications of such networks.
- Nowadays, sensors have been used in e-Health systems. We are interested to apply CA based algorithms for the security of these systems. Cellular automata have been used in designing algorithms for the security of the sensor networks. However, the related problems of this particular application have not been explicitly considered.

In a summary, we are interested to continue designing local algorithms for different optimization problems of wireless sensor networks.

Bibliography

- [1] G. Acs and L. Buttyan. A taxonomy of routing protocols for wireless sensor networks [online]. <http://www.hit.bme.hu/buttyan/publications/AcsB06ht-en.pdf>.
- [2] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3:325–349, 2005.
- [3] K. Akkaya, M. Younis, and M. Bangad. Sink repositioning for enhanced performance in wireless sensor networks. *Computer Networks*, 49(4):512–534, 2005.
- [4] I. Akyildiz, D. Pompili, and T. Melodia. Underwater acoustic sensor networks: Research challenges. *Ad Hoc Networks*, 3:257–279, 2005.
- [5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, November 2002.
- [6] F. Al-Turjman. Grid-based deployment for wireless sensor networks in outdoor environment monitoring applications. *PhD thesis, School of Computing, Queen's University*, 2011.

- [7] S. M. Alam and Z. J. Haas. Coverage and connectivity in three-dimensional underwater sensor networks. *Wireless Communications and Mobile Computing*, 8(8):995–1009, 2008.
- [8] W. Alsalih. Mobile data collectors in wireless sensor networks. *PhD thesis, School of Computing, Queen's University*, 2009.
- [9] W. Alsalih, H. Hassanein, and S. G. Akl. Placement of multiple mobile data collectors in wireless sensor networks. *Ad Hoc Networks*, 8(4):378 – 390, 2010.
- [10] K. M. Alzoubi, P. J. Wan, and O. Frieder. New distributed algorithm for connected dominating set in wireless ad hoc networks. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, HICSS, pages 3849–3855. IEEE, 2002.
- [11] I. Amundson and X. D. Koutsoukos. A survey on localization for mobile wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile entity localization and tracking in GPS-less environments*, MELT'09, pages 235–254, Berlin, Heidelberg, 2009. Springer-Verlag.
- [12] H. Ando, I. Suzuki, and M. Yamashita. Formation and agreement problems for synchronous mobile robots with limited visibility. In *Proceedings of the 1995 IEEE International Symposium on Intelligent Control*, pages 453 –460. IEEE, aug 1995.
- [13] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, and M. Gouda. A line in the sand: A wireless sensor

- network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004.
- [14] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 150–161. ACM, 2003.
- [15] A. J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers*, EC-14(3):394–399, 1965.
- [16] X. Bai, C. Zhang, D. Xuan, J. Teng, and W. Jia. Low-connectivity and full-coverage three dimensional wireless sensor networks. In *Proceedings of the 10th ACM international symposium on Mobile ad hoc networking and computing, MobiHoc '09*, pages 145–154. ACM, 2009.
- [17] L. Barriere, P. Flocchini, E. Mesa-Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. In *Proceedings of the IEEE International Symposium on Parallel Distributed Processing*, pages 1–8. IEEE, may 2009.
- [18] YM Baryshnikov, EG Coffman, and KJ Kwak. High performance sleep-wake sensor systems based on cyclic cellular automata. In *Proceedings of the International Conference on Information Processing in Sensor Networks, IPSN'08.*, pages 517–526. IEEE, 2008.
- [19] I. Benenson and P. M. Torrens. *Geosimulation Automata-based modeling of urban phenomena*. John Wiley & Sons, 2004.

- [20] S.C. Benjamin, N.F. Johnson, and P.M. Hui. Cellular automata models of traffic flow along a highway containing a junction. *Journal of Physics A: Mathematical and General*, 29(12):3119–3127, 1996.
- [21] E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning Ways for Your Mathematical Plays*. Number v. 2. A.K. Peters, 2003.
- [22] E. S. Biagioni and K. W. Bridges. The application of remote sensor technology to assist the recovery of rare and endangered species. *International Journal of High Performance Computing Applications*, 16:2292–2330, 2002.
- [23] M. Cardei and J. Wu. Energy-efficient coverage problems in wireless ad-hoc sensor networks. *Computer communications*, 29(4):413–420, 2006.
- [24] A. Cerpa, J. Elson, M. Hamilton, J. Zhao, D. Estrin, and L. Girod. Habitat monitoring: application driver for wireless communications technology. In *Proceedings of the Workshop on Data communication in Latin America and the Caribbean*, SIGCOMM LA '01, pages 20–41. ACM, 2001.
- [25] P.P. Chaudhuri. *Additive cellular automata: theory and applications*. IEEE Computer Society Press, 1997.
- [26] Q. Chen, A. E. Mynett, and A.W. Minns. Application of cellular automata to modelling competitive growths of two underwater species chara aspera and potamogeton pectinatus in lake veluwe. *Ecological Modelling*, 147(3):253 – 265, 2002.

- [27] X. Cheng, X. Huang, D. Li, W. Wu, and D.Z. Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003.
- [28] S. Choudhury, S. G. Akl, and K. Salomaa. Cellular automaton based algorithms for depth adjustment in underwater sensor networks. In *To appear in Proceedings of the 6th International Workshop on Wireless Sensor, Actuator and Robot Networks, Las Vegas, October 2012*.
- [29] S. Choudhury, S. G. Akl, and K. Salomaa. Cellular automaton based motion planning algorithms for mobile sensor networks. In *To appear in Proceedings of the First International Conference on the Theory and Practice of Natural Computing, Tarragona, Spain, October, 2012*.
- [30] S. Choudhury, S. G. Akl, and K. Salomaa. Energy efficient cellular automaton based algorithms for mobile wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference, WCNC 2012*.
- [31] S. Choudhury, K. Salomaa, and S. G. Akl. A cellular automaton model for connectivity preserving deployment of mobile wireless sensors. In *Proceedings of the 2nd IEEE International Workshop on Smart Communication Protocols and Algorithms, ICC'12 WS - SCPA*.
- [32] S. Choudhury, K. Salomaa, and S. G. Akl. A cellular automaton model for wireless sensor networks. In *Proceedings of the 22nd Second IASTED International Symposium on Modelling and Simulation, MS 2011*.

- [33] S. Choudhury, K. Salomaa, and S. G. Akl. A cellular automaton model for wireless sensor networks. *Journal of Cellular Automata*, 7(3):243–258, 2012.
- [34] A. Clarridge and K. Salomaa. Analysis of a cellular automaton model for car traffic with a slow-to-stop rule. *Theoretical Computer Science*, 411(3839):3507 – 3515, 2010.
- [35] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, volume 4 of *ICRA '04*, pages 3602 – 3608. IEEE, May, 2004.
- [36] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *Transactions on Robotics and Automation*, 20(2):243 – 255, april 2004.
- [37] G. Cowan. *Statistical data analysis*. Oxford science publications. 1998.
- [38] K. Culik, II and S. Dube. An efficient solution of the firing mob problem. *Theoretical Computer Science*, 91(1):57–69, 1991.
- [39] R.O. Cunha, A.P. Silva, A.A.F. Loreiro, and L.B. Ruiz. Simulating large wireless sensor networks using cellular automata. In *Proceedings of the 38th Annual Simulation Symposium*, pages 323–330. IEEE, 2005.
- [40] F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(10):908–920, 2004.

- [41] A.K. Das. Additive cellular automata : Theory and application as a built-in self-test structure. *PhD thesis, I.I.T. Kharagpur, India*, 1990.
- [42] B. Degener, B. Kempkes, P. Kling, and F. Heide. A continuous, local strategy for constructing a short chain of mobile robots. 6058:168–182, 2010.
- [43] B. Degener, B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk, and R. Wattenhofer. A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, SPAA '11, pages 139–148. ACM, 2011.
- [44] C. Detweiler, M. Doniec, M. Jiang, M. Schwager, R. Chen, and D. Rus. Adaptive decentralized control of underwater sensor networks for modeling underwater phenomena. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 253–266. ACM, 2010.
- [45] C. Detweiler, M. Doniec, I. Vasilescu, E. Basha, and D. Rus. Autonomous depth adjustment for underwater sensor networks. In *Proceedings of the Fifth ACM International Workshop on UnderWater Networks*, WUWNet '10, pages 12:1–12:4. ACM, 2010.
- [46] M. Dynia, J. Kutkowski, P. Lorek, and F. Heide. Maintaining communication between an explorer and a base station. In Yi Pan, Franz Rammig, Hartmut Schneck, and Mauricio Solar, editors, *Biologically Inspired Cooperative Computing*, volume 216 of *IFIP International Federation for Information Processing*, pages 137–146. Springer Boston, 2006.

- [47] P. C. Fischer. Generation of primes by a one-dimensional real-time iterative array. *J. ACM*, 12(3).
- [48] G. Fletcher, Xu Li, A. Nayak, and I. Stojmenovic. Randomized robot-assisted relocation of sensors for coverage repair in wireless sensor networks. In *Proceedings of the 72nd Vehicular Technology Conference Fall*, (VTC 2010-Fall, pages 1–5, sept. 2010).
- [49] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005.
- [50] M. Gardner. *Mathematical games: Conway’s game of life*. Scientific American, October 1970.
- [51] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*. WH Freeman and Company, San Francisco, Calif, 1979.
- [52] M. Garzon. *Models of massive parallelism, Analysis of cellular automata and neural networks*. Texts in Theoretical Computer Science, Springer-Verlag, 1995.
- [53] Y. Han and S. Ko. A cellular automaton model for car traffic with a form-one-lane rule. In *Proceedings of the 16th International Conference on Implementation and Application of Automata*.
- [54] F. Heide and B. Schneider. Local strategies for connecting stations by small robotic networks. 268:95–104, 2008.

- [55] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li. Research challenges and applications for underwater sensor networking. In *In Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 228–235, 2006.
- [56] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, volume 2, page 10 pp., January, 2000.
- [57] N. Heo and P. K. Varshney. Energy-efficient deployment of intelligent mobile sensor networks. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35(1):78 – 92, 2005.
- [58] N. Heo and P. K. Varshney. A distributed self spreading algorithm for mobile wireless sensor networks. In *Proceedings of 2003 IEEE Wireless Communications and Networking, 2003*, volume 3 of *WCNC 2003*, pages 1597 –1602,, March, 2003.
- [59] A. Howard, M. J. Mataric, and G. S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems*, pages 299–308, 2002.
- [60] C. Huang, Y. Tseng, and L. Lo. The coverage problem in three-dimensional wireless sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, volume 5 of *GLOBECOM '04*, pages 3182 – 3186. IEEE, November, 2004.

- [61] A. Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific.
- [62] K. Islam. Energy aware techniques for certain problems in wireless sensor networks. *Ph.D. thesis, School of Computing, Queen's University, ON, Canada*, 2010.
- [63] K. Islam and S. G. Akl. A localized algorithm for target monitoring in wireless sensor networks. In *Proceedings of the 8th International Conference on Ad-Hoc, Mobile and Wireless Networks, ADHOC-NOW '09*, pages 391–396. Springer-Verlag, 2009.
- [64] S. Jaber, A. M. Rahmani, and A. K. Zadeh. Trusted data fusion by using cellular automata in wireless sensor networks. In *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference, CEAS '11*, pages 145–151. ACM, 2011.
- [65] K. Jaros, M. Friedhelm, and F. Heide. Optimal strategies for maintaining a chain of relays between an explorer and a base camp. *Theoretical Computer Science*, 410(36):3391 – 3405, 2009.
- [66] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *SIGPLAN Not.*, 37(10):96–107, October, 2002.
- [67] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *ACM SIGOPS operating systems review*, 36(5):96–107, 2002.

- [68] K. Kar and S. Banerjee. Node placement for connected coverage in sensor networks. In *Proceedings of the Modeling and Optimization in Mobile, Ad hoc and Wireless Networks*, volume 3 of *WiOpt*, 2003.
- [69] S. Ko, H. Lee, and Y. Han. An improved cellular automaton model for wireless sensor networks. *Submitted*, 2012.
- [70] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.
- [71] W. Li, A.Y. Zomaya, and A. Al-Jumaily. Cellular automata based models of wireless sensor networks. In *Proceedings of the 7th ACM international symposium on Mobility management and wireless access*, pages 1–6. ACM, 2009.
- [72] X. Li. Improving area coverage by mobile sensor networks. *Ph.D. thesis, SCS, Carleton University, ON, Canada*, 2008.
- [73] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. Strictly localized sensor self-deployment for optimal focused coverage. *IEEE Transactions on Mobile Computing*, 10(11):1520–1533, 2011.
- [74] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. Focused-coverage by mobile sensor networks. In *Proceedings of the IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, MASS 2009, October, 2009.
- [75] C. Y. Lin, W. C. Peng, and Y. C. Tseng. Efficient in-network moving object tracking in wireless sensor networks. *Transactions on Mobile Computing*, pages 1044–1056, 2006.

- [76] B. Liu, P. Brass, O. Dousse, P. Nain, and D. Towsley. Mobility improves coverage of sensor networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing, MobiHoc '05*, pages 300–308. ACM, 2005.
- [77] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, WSNA '02*, pages 88–97. ACM, 2002.
- [78] D. Malan, T. Fulford-jones, M. Welsh, and S. Moulton. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, April, 2004.
- [79] F. B. Manning. An approach to highly integrated, computer-maintained cellular arrays. *IEEE Transaction on Computing.*, 26(6):536–552, 1977.
- [80] M. Min, H. Du, X. Jia, C.X. Huang, S.C.H. Huang, and W. Wu. Improving construction for connected dominating set with steiner tree in wireless sensor networks. *Journal of Global Optimization*, 35(1):111–119, 2006.
- [81] E.F. Moore. *Sequential machines: selected papers*. Addison-Wesley series in computer science and information processing. Addison-Wesley Pub. Co., 1964.
- [82] S. A. Munir, B. Ren, W. Jiao, B. Wang, D. Xie, and J. Ma. Mobile wireless sensor network: Architecture and enabling technologies for ubiquitous computing. In *Proceedings of the 21st International Conference on Advanced Information*

- Networking and Applications Workshops*, volume 2 of *AINAW '07*, pages 113–120, 2007.
- [83] K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *Journal de Physique I*, 2(12):2221–2229, 1992.
- [84] A. Nayak and I. Stojmenovic. *Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication*. Wiley-Interscience, New York, NY, USA, 2010.
- [85] J. Neumann. *The general and logical theory of automata*, pages 1–41. Wiley, Pasadena CA, 1951.
- [86] M. R. Pac, A. M. Erkmen, and I. Erkmen. Scalable self-deployment of mobile sensor networks: A fluid dynamics approach. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1446–1451, oct. October, 2006.
- [87] J. Partan, J. Kurose, and B. N. Levine. A survey of practical issues in underwater networks. In *Proceedings of the 1st ACM international workshop on Underwater networks*, WUWNet '06, pages 17–24. ACM, 2006.
- [88] S. Poduri and G. S. Sukhatme. Constrained coverage for mobile sensor networks. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation, 2004.*, volume 1 of *ICRA '04.*, pages 165 – 171, April, 2004.
- [89] K. Preston Jr., M. J. B. Duff, S. Leviardi, P. E. Norgren, and J. Toriwaki. Basics of cellular logic with some applications in medical image processing. *Proceedings of the IEEE*, 67(5):826 – 856, May, 1979.

- [90] A. Ray and I. Smith. *Introduction to and Survey of Cellular Automata or Polyautomata Theory*. Automata, Languages, Development. North Holland Pub. Co.
- [91] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A: Statistical Mechanics and its Applications*, 231(4):534 – 550, 1996.
- [92] S. Sahni and X. Xu. Algorithms for wireless sensor network. *International Journal of Distributed Sensor Networks*, 1:35–56, 2005.
- [93] P. Sarkar. A brief history of cellular automata. *ACM Computing Survey*, 32(1):80–107, 2000.
- [94] N. J. Savill and P. Hogeweg. Modelling morphogenesis: From single cells to crawling slugs. *Journal of Theoretical Biology*, 184(3):229 – 235, 1997.
- [95] F. Schweitzer, L. Behera, and H. Mhlenbein. Evolution of cooperation in a spatial prisoner’s dilemma. *Advances in Complex Systems (ACS)*, 05(02):269–299, 2002.
- [96] L. Schwiebert, S. K.S. Gupta, and J. Weinmann. Research challenges in wireless networks of biomedical sensors. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, MobiCom ’01, pages 151–165. ACM, 2001.
- [97] R. C. Shah and J. M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Proceedings of the 2002 Wireless Communications and Networking Conference*, WCNC 2002.

- [98] R. Smith. The application of cellular automata to the erosion of landforms. *Earth Surface Processes and Landforms*, 16(3):273–281, 1991.
- [99] E. M. Sozer, M. Stojanovic, and J. G. Proakis. Underwater acoustic networks. *IEEE Journal of Oceanic Engineering*, 25(1):72–83, 2000.
- [100] M. Srivastava, R. Muntz, and M. Potkonjak. Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving environments. In *Proceedings of the 7th annual international conference on Mobile computing and networking, MobiCom '01*, pages 132–138. ACM, 2001.
- [101] J. Suomela. Optimization problems in wireless sensor networks : Local algorithms and local graphs. *PhD thesis, Department of Computer Science, University of Helsinki, Finland*, 2009.
- [102] N. Tamboli and M. Younis. Coverage-aware connectivity restoration in mobile sensor networks. In *Proceedings of the 2009 IEEE international conference on Communications, ICC'09*, pages 17–21. IEEE, 2009.
- [103] E. Tanin, S. Chen, J. Tatemura, and W. Hsiung. Monitoring moving objects using low frequency snapshots in sensor networks. In *Proceedings of the 9th International Conference on Mobile Data Management, 2008, MDM '08*, pages 25–32, April, 2008.
- [104] M. Tatsuno, Y. Nagai, and Y. Aizawa. Rule-dynamical approach to hippocampal network. *Neurocomputing*, 3840:965–971, 2001.
- [105] A. Y. Teymorian, L. Ma, and X. Cheng. Cab: A cellular automata-based key management scheme for wireless sensor networks. In *Proceedings of the 2007*

- IEEE Military Communications Conference, MILCOM 2007*, pages 1–7. IEEE, October, 2007.
- [106] S. Torbey. Towards a framework for intuitive programming of cellular automata. *M.Sc thesis, School of Computing, Queen's University, ON, Canada*, 2007.
- [107] F. Viani, M. Salucci, P. Rocca, G. Oliveri, and A. Massa. A multi-sensor wsn backbone for museum monitoring and surveillance. In *Proceedings of the 2012 6th European Conference on Antennas and Propagation, EUCAP*, pages 51–52, March, 2012.
- [108] L. Žaloudek and L. Sekanina. Increasing fault-tolerance in cellular automata-based systems. In *Proceedings of the 10th international conference on Unconventional computation, UC'11*, pages 234–245. Springer-Verlag, 2011.
- [109] P.J. Wan, K.M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2002*.
- [110] G. Wang, G. Cao, and T. La Porta. Movement-assisted sensor deployment. *IEEE Transactions on Mobile Computing*, 5(6):640–652, June, 2006.
- [111] G. Wang, T. Wang, W. Jia, M. Guo, and J. Li. Adaptive location updates for mobile sinks in wireless sensor networks. *The Journal of Supercomputing*, 47(2):127–145, 2009.

- [112] H. Wang, J. Elson, L. Girod, D. Estrin, K. Yao, and L. Vanderberge. Target classification and localization in habitat monitoring. In *Proceedings of the International Conference on Speech and Signal Processing*, pages II-597–II-600, April, 2003.
- [113] S. Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(12):1 – 35, 1984.
- [114] S. Wongthanavasuu and R. Sadananda. Pixel-level edge detection using a cellular automata-based model. *Advances in Intelligent Systems: Theory and Applications*, 59:343–351, 2000.
- [115] T. Worsch and H. Nishio. Real-time sorting of binary numbers on one-dimensional ca. *CoRR*, abs/1012.0674, 2010.
- [116] S. E. Yacoubi and A. E. Jai. Cellular automata modelling and spreadability. *Mathematical and Computer Modelling*, 36(910):1059 – 1074, 2002.
- [117] F. Ye, A. Chen, S. Lu, and L. Zhang. A scalable solution to minimum cost forwarding in large sensor networks. In *Proceedings of the 10th International Conference on Computer Communications and Networks*, pages 304 –309, 2001.
- [118] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52:2292–2330, 2008.
- [119] N. Yu and X. Ruan. Penicillin fermentation biomass growth model using cellular automata. In *Proceedings of the 5th World Congress on Intelligent Control and Automation*, volume 1 of *WCICA 2004*, pages 216 – 220, June, 2004.

- [120] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in zebranet. In *Proceedings of the 2nd International Conference on Embedded networked sensor systems*, SenSys '04, pages 227–238. ACM, 2004.
- [121] Y. Zou and K. Chakrabarty. Sensor deployment and target localization based on virtual forces. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications*, INFOCOM 2003.