# ALGORITHMS FOR COVERAGE IMPROVEMENT IN A SENSOR NETWORK

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Sunil Reddy Maddi

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

March 2011

Fargo, North Dakota

North Dakota State University

Graduate School

**Title**

**ALGORITHMS FOR COVERAGE**

**IMPROVEMENT IN SENSOR NETWORK**

**By**

**SUNIL REDDY MADDI**

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

# ABSTRACT

Maddi, Sunil Reddy, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, March 2011. Algorithms for Coverage Improvement in a Sensor Network. Major Professor: Dr. Kendall E. Nygard.

Sensors are devices which have the ability to receive and respond to a signal. These sensors, when used as a group, form a sensor network. Sensors in a sensor network can communicate and transmit data. In the early stages of research on sensor networks, only static sensors were used to form a sensor network. As research advanced, a combination of static and mobile sensors was used to form a wireless sensor network instead of just static sensors. The primary advantage of this type of sensor network over a sensor network with static sensors is the ability of mobile sensors to move to a new location in the network to increase the overall area covered by the sensors.

Some concerns in a wireless sensor network are coverage area, energy consumption of the sensors, the ratio of static and mobile sensors to be used in sensor network, and the deployment of sensors in a network. Major research in sensor networks is focused on addressing the issue of coverage area.

The objective of this paper is to design, implement and analyze Most Overlapped First and Highest Coverage Gain algorithms that address the issue of coverage area in a wireless sensor network. Local Spiral Search was used as the base to develop these two algorithms in combination with the Utility Function. Both algorithms were tested, and the results were analyzed with coverage area and change in overlap area as the metrics. Results showed a significant gain in coverage area using both the algorithms, and there was a consistent change in overlap area for a varying number of sensors.

# ACKNOWLEDGEMENTS

This paper would not have been possible without the guidance, help and suggestions of Dr. Kendall E. Nygard who has made available his support in a number of ways. I owe my deepest gratitude to Dr. Saeed Salem, Dr. Simone Ludwig and Dr. Karl Altenburg for serving on the committee. I am indebted to my family and friends for their continuous support.

# TABLE OF CONTENTS

# LIST OF TABLES

Table Page

# LIST OF FIGURES

Figure Page

# 1. INTRODUCTION

In this technological era, the use of sensors is steadily increasing. Sensors come in various types, and their usage varies depending on the conditions for which they are intended. Some common uses of sensors are to measure or detect temperature, pressure, or light. Due to advances in technology, sensors can be used in groups over a specified area or space to perform a specific function. These groups form a sensor network. To increase the effectiveness of these sensor networks, they can be formed with a combination of static and mobile sensors. The primary advantage of this type of sensor network over a sensor network with only static sensors is the ability to move the mobile sensors to enhance their coverage which, in turn, increases the overall coverage provided by the sensor network. The objective of this paper is to develop and test an application, which uses two different algorithms to enhance the sensors' coverage area in a sensor network and to analyze the results obtained from the two algorithms based on different metrics to determine the effectiveness of these algorithms.

Based on the environment, sensor networks can be operated in both a controlled environment, such as a home or automobile, and also in an uncontrolled environment, such as agricultural surveillance and battlefield surveillance in the military. The main idea behind the functioning of the two algorithms is clockwise spiral movement combined with decision-making based on the value of a utility function. The calculation of the utility function is as follows:

Utility Function (UF) = [W1 * (Area Covered/ Maximum Area Covered)] - [W2 * (Number of Moves/ Maximum Number of Moves)],

where W1 and W2 represent the weights of the coverage area and move count, respectively.

Area Covered = $\pi r^2$ - Overlap area for that sensor

where r represents the radius of the sensor.

There are two user interfaces designed in this paper:

- A user interface to enter the required data for generating a sensor network, and
- A user interface to generate a sensor network and also support the execution of the selected algorithm on a generated grid.

The user interface to enter the required data for generating a sensor network is used to gather all the data required for the random generation of a sensor network. Some of the important data that are entered in this interface are the number of static and mobile sensors, the radius of the sensor, the number of test cases and the maximum number of moves.

The user interface to generate a sensor network consists of a network with randomly deployed sensors on it. The data required for random generation of the grid, deployment of sensors, the radius of the sensor and the maximum number of moves are all obtained from the first user interface. In this user interface, the sensors are represented by various colors for easy identification, such as red for static sensors, green for mobile sensors and blue for sensors that are moved from their initial location. The user interface also has options to select either of the two algorithms for implementation on the sensor network that is randomly generated. All the generated values for each test case are recorded in an Excel sheet and are analyzed to determine the effectiveness of the two algorithms based on different metrics.

The arrangement of the chapters is as follows: Chapter 2 has a brief discussion about sensor networks and common sensor coverage issues. Chapter 3 describes the problem definition along with the application interface, the calculation of the Utility Function and the two algorithms. Chapter 4 explains various classes and controls used in the interface along with sample output. Chapter 5 presents Results. The conclusions are addressed in Chapter 6.

# 2. LITERATURE REVIEW

Various applications, such as environmental monitoring, infrastructure management, public safety, health care, home and office security, transportation, and military surveillance, make use of technology from wireless sensor networks [1]. In the initial stages of sensor development, sensors were primarily used to sense or detect; to track a specific target or monitor a specific location. As the technology and research progressed, various breakthroughs made sensors sophisticated and diverse, enabling them for use in various situations. When a group of sensors is used or deployed in a specific location for achieving a specific result, this group forms a sensor network.

The ability to sense with the help of a small node, compute, and communicate are three major functions in a wireless sensor network. Depending on the purpose and use, sensors in wireless sensor networks can communicate between one another, transmit data and also respond based on the occurrence of an incident. In complex environments, real-time, large-scale data processing can be facilitated using wireless sensor networks [2]. Many wireless sensor networks used in real world environments make use of these three functions of a wireless sensor network to perform various tasks.

Early research focused on sensor networks where all the sensors in that network were static sensors. As technology advanced and the use of sensors in various fields increased, research was focused on a new breed of wireless sensor networks involving both static and mobile sensors. These types of wireless sensor networks have a significant advantage over wireless sensor networks that only contain static sensors. This is one of the points that is implemented in wireless sensor networks used in this paper. The basic advantage of using both static and mobile sensors in a wireless sensor network is mobile

sensors have the flexibility to change their positions to serve various purposes, such as decreasing the overlap between sensors, increasing the coverage area and consuming energy efficiently. The use of wireless sensor networks varies a great deal based on factors such as the type of environment, number of sensors, the deployment of the sensors in the network and energy consumption. In wireless sensor networks where energy is the primary concern, the emphasis is on designing a better routing, power management or data dissemination protocols [3].

Sensors can be used either in a controlled environment or in an uncontrolled environment. In a controlled environment, sensors can be adjusted based on the needs with their position and energy consumption; also, their communication with other sensors in the network can be monitored. Most sensors used for household appliances and automobiles are examples of sensors used or deployed in a controlled environment. In an uncontrolled environment, sensors are deployed randomly, and several factors affect the overall efficiency of the sensors in an uncontrolled sensor network. Nuclear power plants, battlefield surveillance and military operations are examples of sensors deployed in an uncontrolled environment [4].

Some major issues in a wireless sensor network involve coverage area, energy consumption and tracking. Sensor networks pose a number of challenging conceptual and optimization problems, such as location, deployment, and tracking. One of the fundamental problems in sensor networks is the calculation of coverage [5].

A majority of the research related to addressing the coverage area issue proposed different solutions based on factors such as the object to be covered, deployment

mechanism of the sensors as well as using network connectivity as a metric to address the coverage-area issue [6].

One way of conserving energy in a wireless sensor network is through switching off some redundant nodes in the sensor network. This type of routine is possible only in a controlled environment where sensor placement is done in a precise manner with coverage area as a primary concern. This point of conserving the energy of a sensor in a wireless sensor network is also considered while developing the algorithms using Local Spiral Search and Utility Function. The Utility Function keeps track of number of moves of a mobile sensor and also the user has the option to limit the number of moves based on importance given to conserving energy in sensor network.

Another issue that is often seen in wireless sensor networks is deployment of sensors. In this paper both algorithms were applied on sensor networks where the deployment of sensors is done randomly. This is done to simulate real world problems that occur in sensor networks when sensors are randomly deployed like overlapping of sensors and decrease in coverage area due to overlap.

Some of the other issues that occur in wireless sensor networks is the process of communication between the sensors. These communication issues in wireless sensor network also affect the energy of a sensor and there by overall efficiency of a sensor network.

Most Overlapped First and Highest Coverage gain algorithms in this paper try to address the coverage area in a sensor network along with efficient energy consumption for sensors by decreasing the number of moves mobile sensors make. The basic idea behind both algorithms is a Local Spiral Search [7] in combination with the Utility Function value.

The use of Utility Function is done primarily for the purpose of decision making when the overlapped sensor makes a move on the grid. Both algorithms are tested based on different metrics, and the results gathered from various tests support the fact that there is a significant gain in coverage area and also a consistent decrease in overlap area.

# 3. PROBLEM DEFINITION IN SENSOR NETWORKS

Most Overlapped First Algorithms and Highest Coverage Gain First Algorithms described in this paper are designed using the Spiral Search Algorithm as base. These algorithms are then customized, and additional factors are added to enhance the effectiveness of the two algorithms. This chapter deals with the problem definition and explains the two algorithms' functionality in detail. Subsections 3.1, 3.2 and 3.3 deal with problem definition, assumptions made and exceptions that are known. The Local Spiral Search Algorithm, Most Overlapped First Algorithm and Highest Coverage Gain First Algorithm are explained in Subsections 3.4, 3.5 and 3.6, respectively.

## 3.1. Problem Definition with Respect to Coverage Area in a Sensor Network

One of the problems in a sensor network with randomly deployed sensors is effective area coverage by those sensors. In many instances when the sensors are randomly deployed to be covered, sensor coverage becomes overlapped. The overlapped sensor coverage in turn, results in less area covered by those sensors than their actual capacity. One solution to this problem is to use a combination of static and mobile sensors instead of just static sensors. The main advantage with this process is that mobile sensors can be moved from their initial position when overlap occurs. The problem in this scenario is what pattern to follow when moving a mobile sensor to enhance coverage. The Most Overlapped First Algorithm and Highest Coverage Gain First Algorithm that will be discussed try to address this problem.

The two algorithms are tested with a user interface which demonstrates their functionality and also generates the test results required for analyzing the algorithms' effectiveness. The user interface is developed using Microsoft's .NET framework, and the entire application is a Windows-based application. The main advantage with Microsoft's .NET Framework is the number of languages it supports in addition to the ease of designing the user interface with various features provided by the framework. The code for functionality and several events in the user interface are done in the C# language. The following section describes the various assumptions.

## 3.2. Assumptions

The first assumption we would make for this algorithm is that the sensor can sense the next adjacent cell to its right to which it is about to move in the spiral path; what we mean by sense is that the sensor will know the value of the area covered if it moves to that position. Based on this assumption, we will calculate the Utility Function (Moving) value. In the next step, we compare the Utility Function (Moving) and Utility Function (Not Moving) values.

The second assumption is that the maximum number of moves a mobile sensor can make in a spiral path is 8, which will be the maximum value that a user can input. The user also has the option of entering a number less than 8, but the maximum allowed value is 8. This assumption is made to serve two purposes. First, it keeps a limit on the number of moves a mobile sensor can make in the spiral path, and second, due to limit in the number of moves helps conserve the energy of the mobile sensor in the sensor network.

## 3.3. Exceptions

For the sensor, the generated grid is like a wall. Grid here represents the geographical area where the sensors are randomly deployed. Sensors randomly deployed cannot cross the grid. This assumption is made to deal with sensor overlap along the grid end lines, which represent the border of the geographical area to be covered. Due to this assumption for sensor overlap at the grid border, the number of available moves is limited or there are no moves, depending on its location. For example, in the Figure 3.1, the top line indicates the wall of the grid in a sensor network.

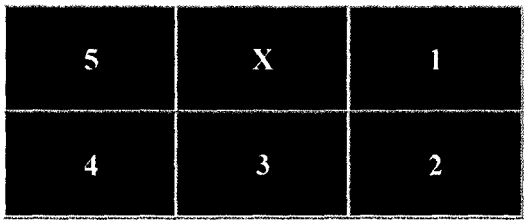

Figure 3.1. Sample Grid for Local Spiral Search Exception

Suppose we have entered the maximum number of moves as 8 in the Sensor Data Form. In Figure 3.1, say that X is the position of the overlapped sensor and that the thick line represents the end line of the generated grid. In such cases, there is no guarantee that the overlapped sensor will have the maximum number of moves that the user entered in the Sensor Data Form. In the Figure 3.1 case, the sensor only has 5 moves that it can make from its initial position.

## 3.4. The Local Spiral Search Algorithm

This subsection explains the Local Spiral Search Algorithm that forms the basis for the Most Overlapped First and Highest Coverage Gain algorithms. When a combination of static and mobile sensors is deployed randomly, some sensor coverage overlaps. This overlap results in less covered area that, in turn, decreases the overall efficiency of the

sensor network. To counter such cases, the Local Spiral Search Algorithm can be used to move the overlapped mobile sensors to a new position. In a Spiral Search, the sensor movement is in a clockwise direction.

For example, please consider Figure 3.2 that represents a part on the grid. Let us say X is the initial position of the overlapped sensor. According to Spiral Search, the sensor moves outward in a spiral path in the clockwise direction. From the initial position, X, the sensor moves to position 1 which is the first move towards its right and, from there, on to 2 which is one move down and next to 3 which is one move towards its left, etc. until the sensor finds a suitable position to enhance the coverage area.

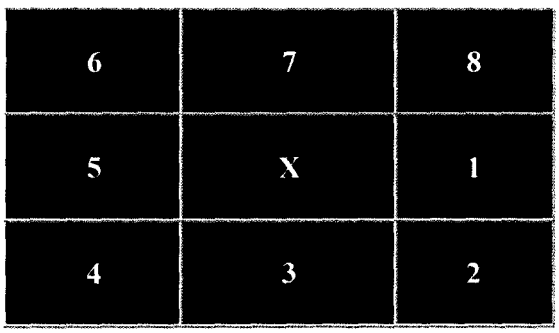

Figure 3.2. Sample Grid for Local Spiral Search

This technique can be applied to sensors in a sensor network in the following scenarios:

- When there is overlap between two mobile sensors in a sensor network. In this case, one of the sensors can be moved to achieve a coverage gain; and
- When there is overlap between a static and mobile sensor. In this case, the mobile sensor is moved to a new location.

Using Local Spiral Search, the mobile sensors can move to the surrounding cells, thereby increasing the coverage area which, in turn, increases the overall efficiency of the sensor network. In addition to Local Spiral Search, the Utility Function is used to decide on the movement of overlapped sensors. The combination of Spiral Search and the Utility

Function further improves the effectiveness of this routing pattern. The basic formula of the Utility Function is as follows:

Utility Function (UF) = [W1 * (Area Covered/Maximum Area Covered)] - [W2 * (Number of Moves/Maximum Number of Moves)],

where W1 and W2 represent the weights of coverage area and move count, values entered in Sensor Data form respectively.

Maximum Number of Moves = Value entered in Sensor Data form

Area Covered = $\pi r^2$- Overlap area for that sensor

Where r represents the radius of the sensor

Maximum Area Covered = $\pi r^2$

Number of Moves = Value of the moves made by the mobile sensor

Based on the above Utility Function formula, we start the movement by calculating the Utility Function values for moving and not moving. Refer to Figure 3.3; let us say that X is the initial position of the most overlapped sensor.

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

Figure 3.3. Sample Grid for Utility Function

For the first move, the Utility Function at position X refers to UF (Not Moving) while 1 represents UF (Moving). The one assumption we would be making in this algorithm is that the sensor can sense the next cell to which it is about to move in the spiral path; what we

mean by sense is that the sensor will know the value of the covered area if it moves to that position. Based on this assumption, we will calculate the Utility Function (Moving) value. In the next step, we compare both the Utility Function (Moving) and Utility Function (Not Moving) values. Based on the value, here are the possible cases:

- If the Utility Function (Moving) value is less than the Utility Function (Not Moving) value, then the sensor does not move. It stays at its original position.
- If the Utility Function (Moving) value is greater than or equal to the Utility Function (Not Moving) value, then the sensor moves to position 1.

Again at position two, the Utility Function value is evaluated, and based on the value of Utility Function of moving and not moving, the decision is made about whether to move forward or to stop. This process is repeated for all the overlapped sensors in the sensor network. This process of evaluating a mobile sensor's moves using the Utility Function helps in two ways. First, it prevents the mobile sensor from making unnecessary moves which, in turn, conserves the energy of the sensor; there by helps in achieving an efficient sensor network by increasing the coverage gain. An overlapped mobile sensor follows the following steps while evaluating a Utility Function:

STEP 1: FOR THE SELECTED OVERLAPPED SENSOR, CALCULATE THE UTILITY FUNCTION VALUE AT THE CURRENT LOCATION KNOWN AS THE UTILITY FUNCTION OF NOT MOVING USING THE FORMULA AS STATED BELOW:

UF (NOT MOVING) = [W1 * (Area Covered/Maximum Area Covered)] - [W2 * (Number of Moves/Maximum Number of Moves)]

STEP 2: CALCULATE THE UTILITY FUNCTION VALUE FOR THE NEXT LOCATION TO WHICH THE SENSOR IS SUPPOSED TO MOVE, KNOWN AS THE UTILITY FUNCTION OF MOVING BY USING THE FORMULA STATED BELOW:

UF (MOVING) = [W1 * (Area Covered/Maximum Area Covered)] - [W2 * (Number of Moves/Maximum Number of Moves)]

STEP 3: IF THE VALUE OF THE UTILITY FUNCTION (MOVING) > UTILITY FUNCTION (NOT MOVING)

AND

STEP 4: IF (NUMBER OF MOVES < MAXIMUM NUMBER OF MOVES)

THEN

STEP 5: MOVE TO THE NEXT LOCATION

ELSE

STEP 6: STAY AT THE CURRENT LOCATION

STEP 7: IF (MOVE TO THE NEXT LOCATION)

STEP 8: REPEAT STEPS 1 TO 6

ELSE

STEP 9: STOP

Figure 3.4 depicts the flowchart of a sensor with decision to move based on the Utility Function value. The flowchart in Figure 3.1 explains how an overlapped mobile sensor is evaluated at each position before it is either moved to a new location or stay in the current location in the spiral path. This process of evaluating the overlapped mobile sensor remains the same in both algorithms. The only thing that differs in both the algorithms is

the order in which the overlapped sensors are selected in a wireless sensor network. All the overlapped mobile sensors follow these events when moving from one location to another in a sensor network. The following subsection will discuss the Most Overlapped First Algorithm in detail

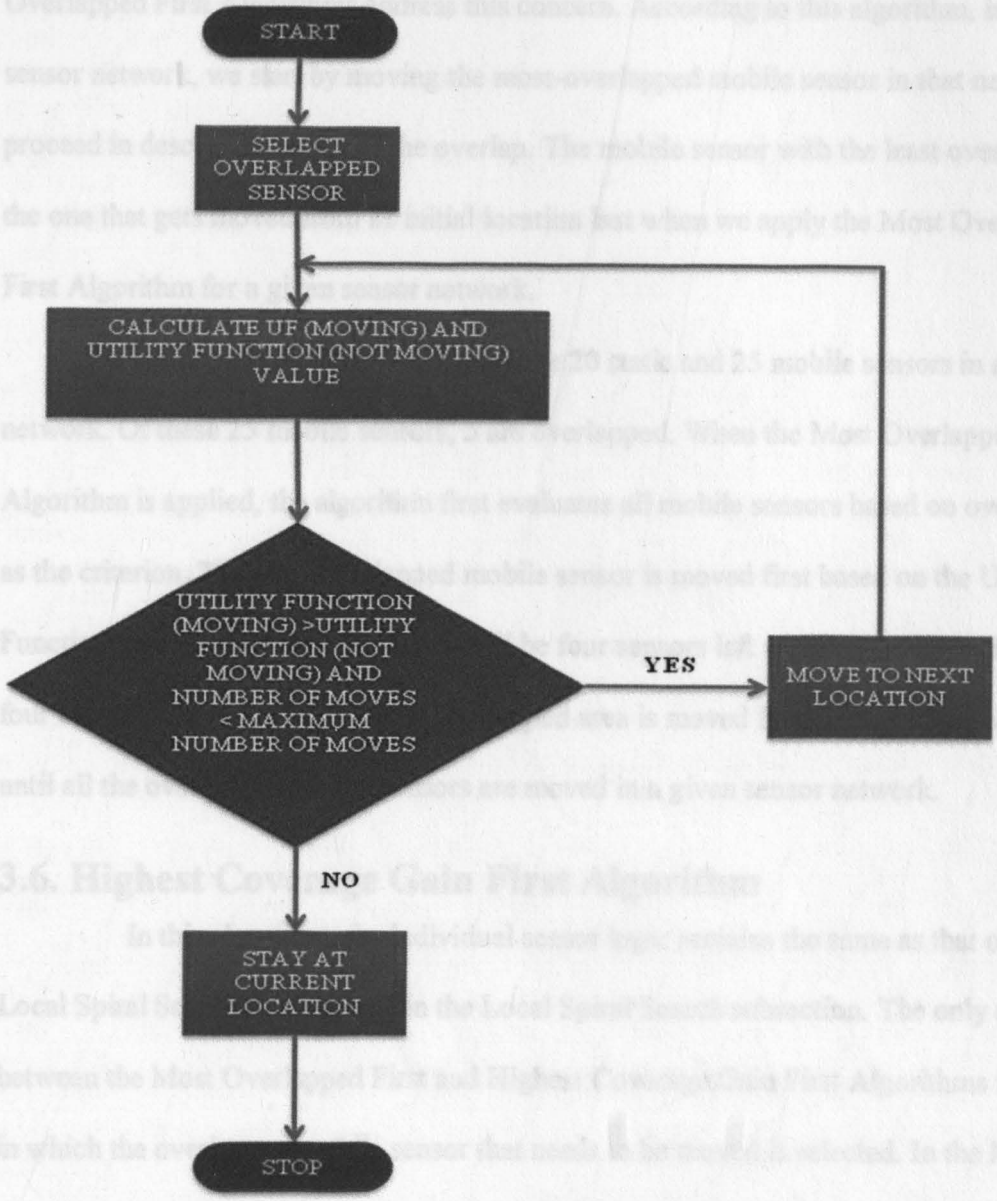


Figure 3.4. Flowchart for a Mobile Sensor Based on Utility Function Evaluation

## 3.5. Most Overlapped First Algorithm

In this algorithm, the individual sensor logic remains the same as that of the Local Spiral Search explained previously. The main point that needs to be addressed while applying the Local Spiral Search along with the Utility Function is which sensor needs to be moved and what process needs to be followed for the selecting the sensors. Most Overlapped First Algorithms address this concern. According to this algorithm, in a given sensor network, we start by moving the most-overlapped mobile sensor in that network and proceed in descending order of the overlap. The mobile sensor with the least overlap area is the one that gets moved from its initial location last when we apply the Most Overlapped First Algorithm for a given sensor network.

Let us take the case where we have 20 static and 25 mobile sensors in a sensor network. Of these 25 mobile sensors, 5 are overlapped. When the Most Overlapped First Algorithm is applied, the algorithm first evaluates all mobile sensors based on overlap area as the criterion. The most overlapped mobile sensor is moved first based on the Utility Function value in a spiral path. There will be four sensors left in this case. Even from these four sensors, the one with the most overlapped area is moved first. This process continues until all the overlapped mobile sensors are moved in a given sensor network.

## 3.6. Highest Coverage Gain First Algorithm

In this algorithm, the individual sensor logic remains the same as that of the Local Spiral Search as explained in the Local Spiral Search subsection. The only difference between the Most Overlapped First and Highest Coverage Gain First Algorithms is the way in which the overlapped mobile sensor that needs to be moved is selected. In the Highest Coverage Gain Algorithm, the overlapped mobile sensor that, on its first move, will give the highest coverage gain is selected, and it is moved first. This process is continued for all

the overlapped mobile sensors in the descending order of their coverage gain on the first move.

The only difference between Most Overlapped First and Highest Coverage Gain algorithms is the way in which the overlapped mobile sensors that need to be moved are selected. In the Most Overlapped First Algorithm, we move the most overlapped mobile sensor first and proceed in descending order, whereas in the Highest Coverage Gain Algorithm, we first move the overlapped mobile sensor which will give the highest coverage gain and proceed in descending order.

# 4. METHODOLOGY AND CLASSES

The application developed as a part of this project is a combination of two user interfaces. The first user interface, known as Sensor Data, receives input from the user for various parameters required to generate a sensor network along with some additional details, such as weights used in calculating the Utility Function and number of test cases. Once the values entered in the first user interface are validated and submitted, a sensor network is generated on another user interface known as the Generated Sensor Network. The user has the option to select one of the two algorithms to apply for the Generated Sensor Network. Once an algorithm is applied to a sensor network, all the details, such as coverage area, non-coverage area, overlap area and coverage gain, are logged into an Excel file in .xls format. The two User Interfaces under discussion are a combination of various classes and methods which will be discussed briefly in this chapter.

## 4.1. IMovingAlgorithm Class

The IMovingAlgorithm class is the base class for all functionality in the project. Table 4.1 lists the methods that are implemented in the MovingAlgorithmBase class.

Table 4.1. IMovingAlgorithmBase Class: Methods

| RETURN TYPE | METHOD | PURPOSE |
|---|---|---|
| Void | improveSensor () | This method provides functionality for determining best position for a sensor and to move sensor to that position |
| MobileSensor | getBestCandidate () | This method finds the overlapped sensor based on algorithm selected |
| Position | getBestPosition () | This method finds the best position for an overlapped sensor when moved in spiral path |

Both algorithms implement this interface class. The methods declared in this class are implemented in the MovingAlgorithmBase class that inherits the IMovingAlgorithm class.

### 4.1.1. MovingAlgorithmBase Class

This class implements all the methods that were declared in IMovingAlgorithm class. The main purpose of this class is as follows:

- To select a suitable sensor, depending on the algorithm selected by the user, and
- To move the sensor based on the utility function value.

The below-listed methods find the sensor, move it from its initial location on the grid by comparing the utility function value and check to see if the final position is the best position for that particular sensor. Table 4.2 lists the methods that are implemented in the MovingAlgorithmBase class.

Table 4.2. MovingAlgorithmBase Class: Methods

| RETURN TYPE | METHOD | PURPOSE |
|---|---|---|
| Void | improveSensor () | This method provides functionality for determining best position for a sensor and to move sensor to that position |
| MobileSensor | getBestCandidate () | This method finds the overlapped sensor based on algorithm selected |
| Position | getBestPosition () | This method finds the best position for an overlapped sensor when moved in spiral path |

### 4.1.2. MostOverlappedFirstAlgorithm Class

This class is called when the user selects the option to implement the Most Overlapped First Algorithm. Table 4.3 shows the method implemented in the MostOverlappedFirstAlgorithm class. The main functionality of this class is to find the most overlapped sensor on the sensor network and return it to the MovingAlgorithmBase

class to implement the various methods required for moving a sensor from one location to another.

Table 4.3. MostOverlappedFirstAlgorithm Class: Method

| RETURN TYPE | METHOD | PURPOSE |
|---|---|---|
| MobileSensor | getMoreOverlappedMobileSensor () | This method returns the most overlapped sensor present on the sensor network when this method is called. |

### 4.1.3. HighestCoverageGainAlgorithm Class

This class is called when the user selects the option to implement the Highest Coverage Gain Algorithm. Table 4.4 shows the methods implemented in the HighestCoverageGainAlgorithm class. The main functionality of this class is to find the sensor which, on the first move, will give the highest coverage gain of all available sensors on the sensor network and return it to the MovingAlgorithmBase class to implement the various methods required for moving a sensor from one location to another.

Table 4.4. HighestCoverageGainAlgorithm Class: Methods

| RETURN TYPE | METHOD | PURPOSE |
|---|---|---|
| MobileSensor | getMoreImprovingMobileSensor() | This method finds the candidate sensor which will provide the highest coverage gain with a move. |
| Double | calcImprovement() | This method calculates the intersection area for a mobile sensor. |

## 4.2. Coverage Class

This class stores all the area-related values: total, coverage and overlap area. Table 4.5 shows the variables used in the Coverage class.

Table 4.5. Coverage Class: Variables

| VARIABLE | TYPE | PURPOSE |
|---|---|---|
| Total | Double | Store the value of the total area calculated |
| coverage | Double | Store the value of the coverage area calculated |
| Overlap | Double | Store the value of the overlap area calculated |

This class encapsulates and keeps all data needed to calculate the coverage from move to move and allows the methods to return all the data without having to pass the references. Using the values stored in this class, non-coverage area is calculated. All these values are updated dynamically on the user interface as soon as a sensor makes a move using one of the two algorithms. This class is also used in calculations involving the utility function because coverage area is one parameter of the utility function along with move count.

## 4.3. ClockwiseSpiral Class

This class generates new points from an original point following a clockwise spiral pattern whenever a sensor makes a move. This class has variables which hold values for sensor area, cell area, radius of the sensor and direction of the sensor. This class is called whenever a sensor makes a move on the grid. This class plays a prominent role in the movement of an overlapped sensor on the grid. The directions variable in this class holds the set of directions that needs to be implemented for clockwise spiral movement of a mobile sensor. Table 4.6 shows the variables that are used in this class.

Table 4.6. ClockwiseSpiral Class: Variables

| VARIABLE | TYPE | PURPOSE |
|---|---|---|
| Position | Position | Stores the current position of the sensor |
| sensorArea | Dimension | Stores the sensor-area limits |
| Cell | Dimension | Stores the size of a single cell in the generated grid |
| directions | Directions[] | Holds the pattern of directions to implement the clockwise movement |
| Radius | Int | Value of the sensor radius |
| nextDirection | Int | Holds the direction value for the next movement |

## 4.4. Sensor Data Form

The Sensor Data form contains various input fields which allow the user to enter the values for different parameters required for generating a sensor network. Figure 4.1 shows the user interface of the .NET-based implementation of the Sensor Data form. The User Interface has the ability to store user-entered values and to use them in generating a sensor network.

The Sensor Data form holds values for different parameters as follows:

- Number of Test Cases: This field takes an integer value, and the field is validated using JavaScript validation. The value in this field is the number of sensor networks that can be randomly generated and executed keeping all the remaining values constant. For example, if 5 is entered in the input field, then the user has the option to generate 5 sensor networks, one after the other.
- Radius: This field holds the value for the radius of the sensor coverage for sensors randomly deployed on the Generated Sensor Network. This field takes an integer

value and is validated using JavaScript validation. The user also has the option to select sensor coverage areas of the same size or different sizes. In the case of sensors with different coverage areas, a pop-up window will appear and will allow the user to enter different radii for different sensors.

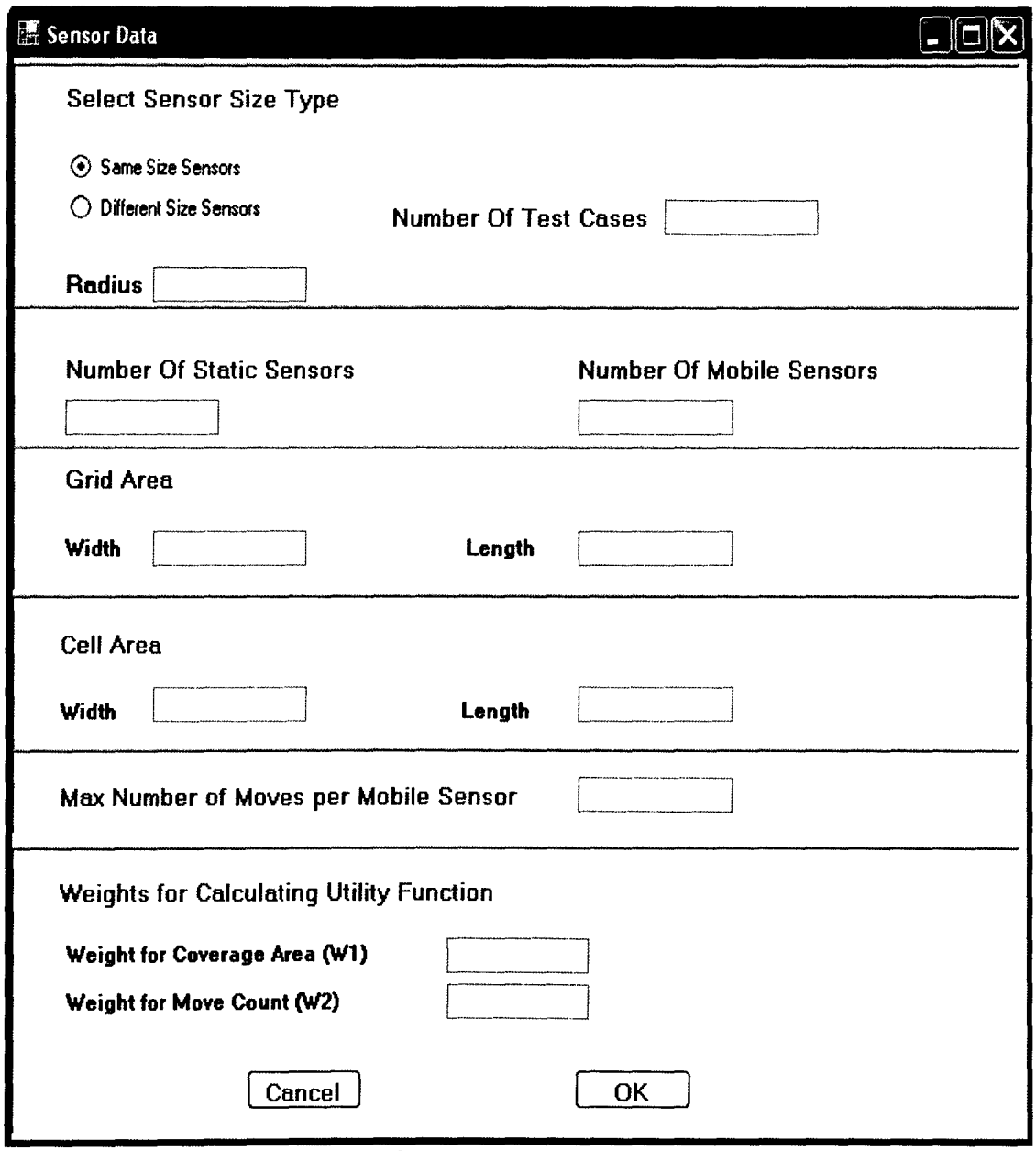

Figure 4.1. Sensor Data User Interface

- Number of Static and Mobile Sensors: This field holds the value for the number of static and mobile sensors that will be deployed randomly on the generated sensor network. These fields take an integer value and are validated using JavaScript validation.
- Grid Area: This field takes two values for width and height, respectively, to generate a grid. These fields take an integer value and are validated using JavaScript validation. For example, if the width and height are entered as 500 and 400, respectively, then a grid is generated in the second user interface for the product of those two values. The combination of these two values decides whether a square or rectangular grid is generated on the second user interface.
- Cell Area: Cell area is the area of the individual cell on the grid. This field takes two values for width and height, respectively. These fields take an integer value and are validated using JavaScript validation. For example, if the width and height are entered as 500 and 400, respectively, for grid area and 50 and 40 for cell area, then a grid with 100 cells will be formed.
- Maximum Number of Moves per Mobile Sensor: The user has the option to specify the maximum number of moves that each overlapped mobile sensor can make in the spiral path on the grid. This field accepts values between 1 and 8, with 8 being the maximum value that it will accept. The field is validated using JavaScript validation, and any attempt to enter a value higher than 8 results in a pop-up error message being displayed.
- Weights for Calculating Utility Function: The user has the option to enter weights for calculating a utility function. The utility function requires two

weights, W1 and W2, with one for coverage area and another for move count. The sum of these two weights is always equal to one. The two fields are validated in such a way that, if the user enters a value in one of the fields, the other input field is automatically populated with the remaining value. For example, let us say the user has entered 0.6 in the W1 field; then, the W2 field will be automatically populated with 0.4. If the user enters a value greater than 1 in W1 field an error message pops up terming the value as invalid.

Apart from the discussed input fields, two buttons, "Cancel" and "OK," are also present on this form. The function of the "Cancel" button is to reset all the values entered in the input fields, and clicking the "OK" button triggers a flow of events: call to generate a grid, deploy static and mobile sensors randomly onto the grid, and populate the text fields with relevant details on the User Interface.

## 4.5. Generated Sensor Network Form

Once all fields are validated with the Sensor Data form and the form is submitted by clicking the "OK" button, the Generated Sensor Network form opens. Figure 4.2 shows the default loading of the Generated Sensor Network form. Before the Generated Sensor Network form loads, a few events occur in the background. The sequences of events that occur are as follows:

- A grid is generated based on the values entered in the grid and cell area fields of the Sensor Data form. For example, if a user enters 500 in both fields of the grid area and 50 in the fields for cell area, then a 10 * 10 grid is generated. This grid will have 100 cells;
- Static and mobile sensors are randomly deployed onto the grid; and

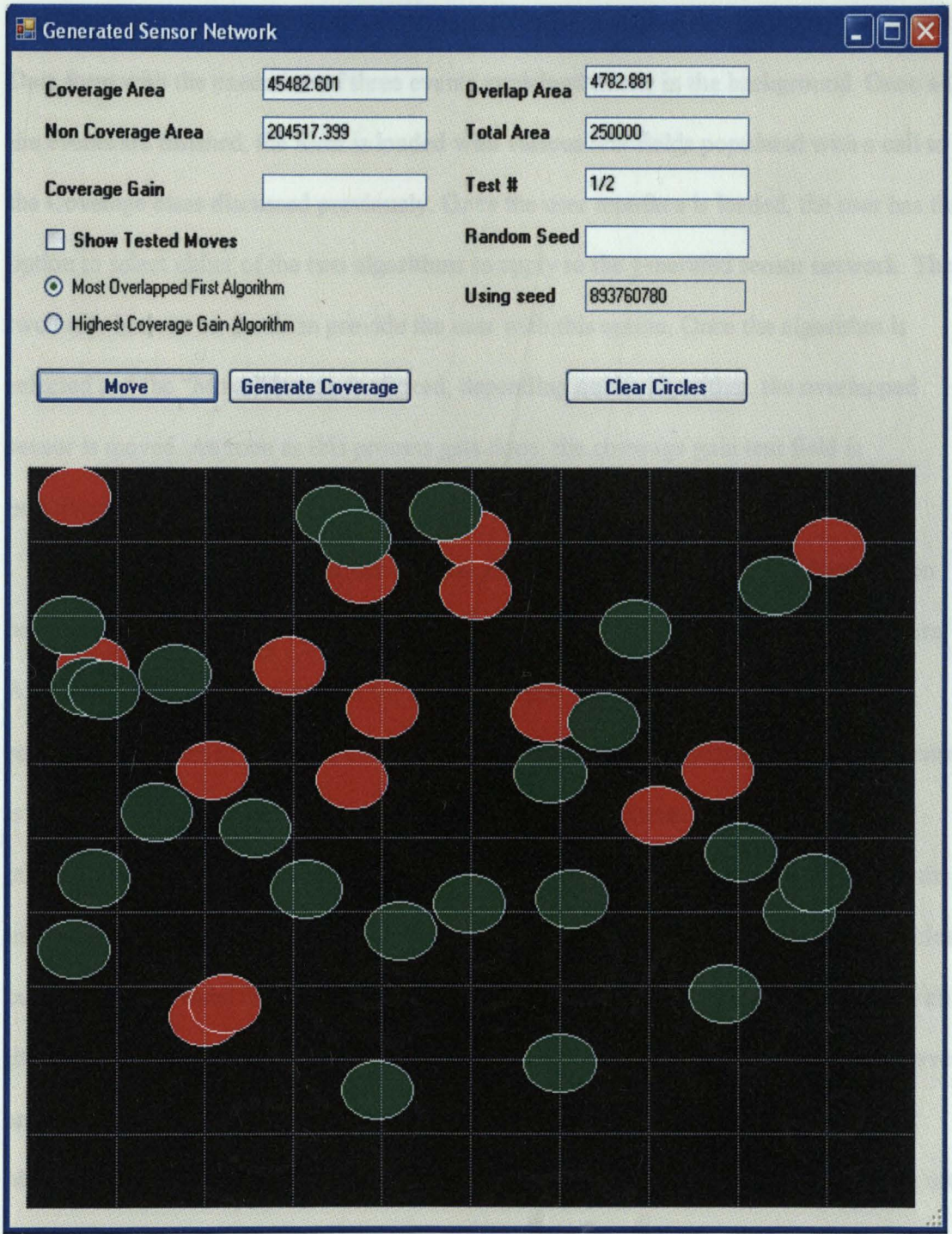


Figure 4.2. Generated Sensor Network User Interface

- Coverage class populates the text fields with relevant values for different areas in calculation along with coverage gain.

The Generated Sensor Network form is loaded by clicking the "OK" button on the Sensor Data form with the execution of three events explained above in the background. Once all the events are finished, the form is loaded with various text fields populated with a call to the Coverage class discussed previously. Once the user interface is loaded, the user has the option to select either of the two algorithms to apply to the generated sensor network. The two radio buttons on the form provide the user with this option. Once the algorithm is selected and the "Move" button is clicked, depending on the algorithm, the overlapped sensor is moved. As soon as this process gets done, the coverage gain text field is populated.

Because a sensor network has many sensors on it, for easy differentiation between static and mobile sensors, they are represented on the grid using different colors. A red color on the grid indicates a static sensor while a green color indicates a mobile sensor. Once a sensor is moved from its initial location to a new position, the moved sensor is represented using a blue color; a yellow, dotted circle represents the moved sensor's initial location. To avoid confusion while running the algorithm on the User Interface, the user is provided with the option to clear the yellow, dotted circles using the "Clear Circles" button. "Show Tested Moves" is a check box on the user interface which allows user with the option to represent the moves tested by an overlapped mobile sensor. The tested moves are represented by solid yellow circle on the grid. When the user selects "Show Tested Moves" checkbox, the yellow circles appear for every overlapped sensor. For clarity at any given point, the yellow circles appear for the last sensor that is moved. Figure 4.3 shows the view of the Generated Sensor Network form after some moves are made using one of the algorithms.

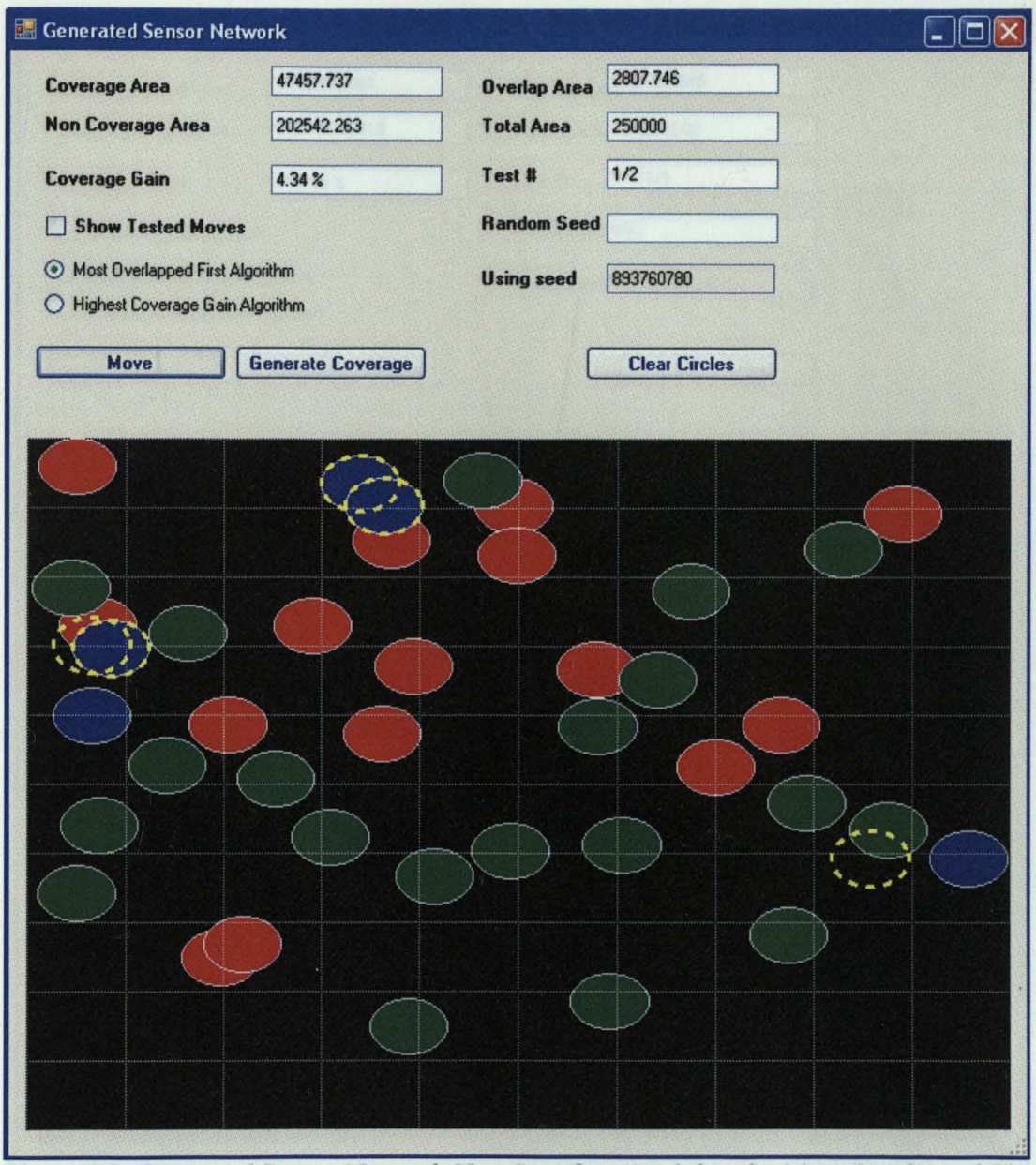


Figure 4.3. Generated Sensor Network User Interface Applying the Most Overlapped First Algorithm

Once the algorithm makes all the moves possible in a particular test case, a pop-up message is displayed and indicates that no more moves are possible in that particular test case. Figure 4.4 shows the Generated Sensor Network form when a pop-up message is displayed to the user.

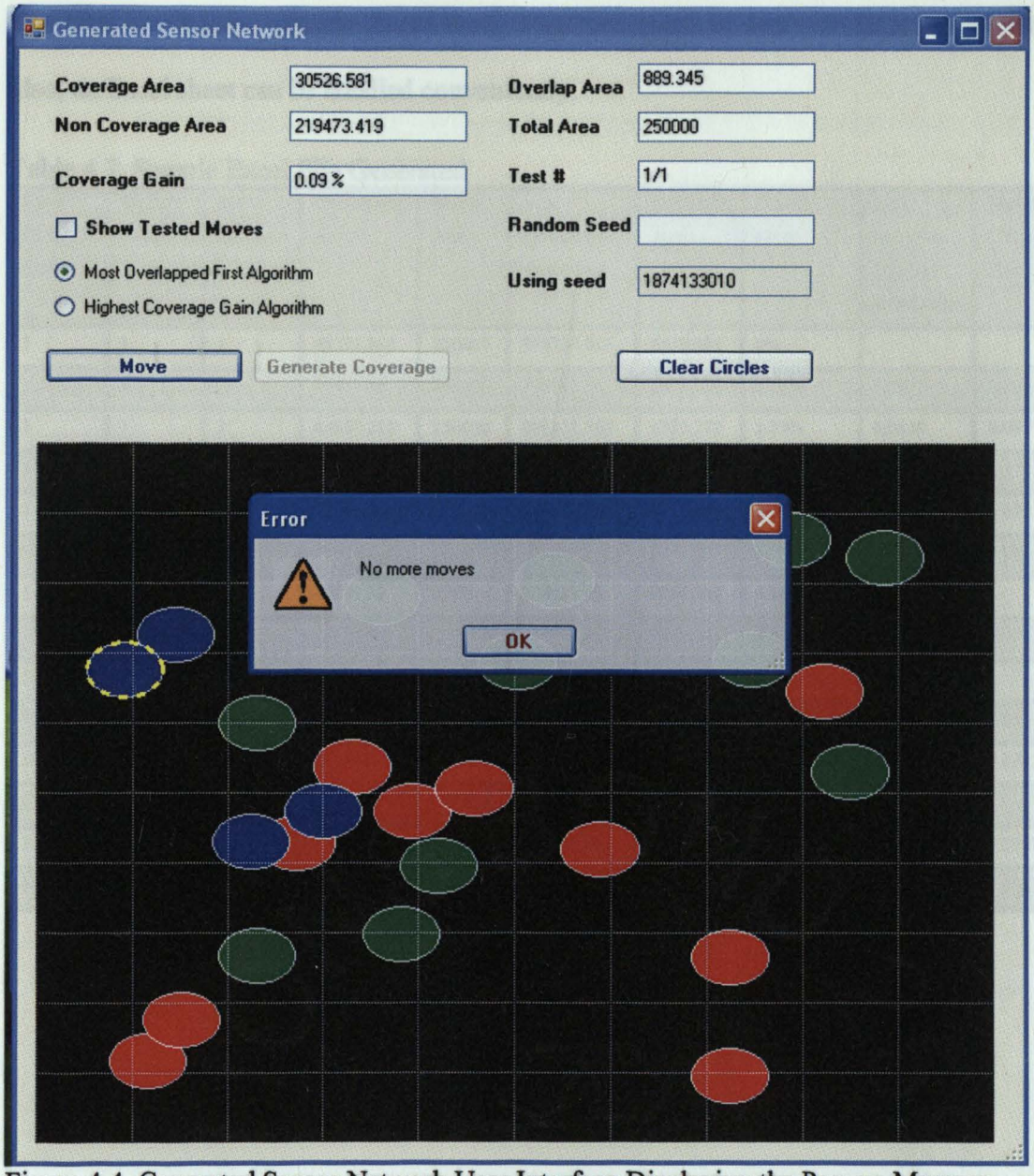


Figure 4.4. Generated Sensor Network User Interface Displaying the Pop-up Message

## 4.6. Sample Output

Once all the test cases are executed using one of the two algorithms, an Excel sheet is generated to store all moves that were made on the sensor network using that algorithm. If the user executes two algorithms, then in two Excel sheets will be generated. Table 4.7 shows a data sample which is stored in the generated Excel sheet in .xls format.

The main idea behind using an Excel sheet is the ease of use to deal with large sets of data; also, an Excel sheet can be handled conveniently.

Table 4.7. Sample Excel File Generated

| Test Case Number | Sensor Number | Move Number | Coverage Area | Total Area | Non-Coverage Area | Overlap Area | Coverage Gain | Not Moving Utility Function | Moving Utility Function |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 43265.988 | 250000 | 206734.012 | 6999.494 | 0% | | |
| 1 | 1 | 2 | 44457.86 | 250000 | 205542.14 | 5807.622 | 2.75% | 0.0865 | 0.0889 |
| 1 | 1 | 3 | 43867.211 | 250000 | 206132.789 | 6398.272 | 1.39% | 0.0889 | 0.0877 |
| 1 | 2 | 1 | 44457.86 | 250000 | 205542.14 | 5807.622 | 2.75% | | |
| 1 | 2 | 2 | 45508.35 | 250000 | 204491.65 | 4757.132 | 5.18% | 0.0889 | 0.091 |
| 1 | 2 | 3 | 44811.385 | 250000 | 205188.615 | 5454.098 | 3.57% | 0.091 | 0.0896 |
| 1 | 3 | 1 | 45508.35 | 250000 | 204491.65 | 4757.132 | 5.18% | | |
| 1 | 3 | 2 | 45131.54 | 250000 | 204868.46 | 5133.943 | 4.31% | 0.091 | 0.0903 |
| 1 | 4 | 1 | 45508.35 | 250000 | 204491.65 | 4757.132 | 5.18% | | |
| 1 | 4 | 2 | 46143.251 | 250000 | 203856.749 | 4122.231 | 6.65% | 0.091 | 0.0923 |
| 1 | 4 | 3 | 46084.849 | 250000 | 203915.151 | 4180.634 | 6.52% | 0.0923 | 0.0922 |
| 1 | 5 | 1 | 46143.251 | 250000 | 203856.749 | 4122.231 | 6.65% | | |
| 1 | 5 | 2 | 46443.249 | 250000 | 203556.751 | 3822.234 | 7.34% | 0.0923 | 0.0929 |
| 1 | 5 | 3 | 45486.119 | 250000 | 204513.881 | 4779.363 | 5.13% | 0.0929 | 0.091 |

# 5. RESULTS

This chapter details the various tests performed with the two algorithms. The performance of the algorithms is compared based on two different metrics, coverage gain and overlap area.

## 5.1. Test Case 1: Most Overlapped First Algorithm

Tests are performed using the Most Overlapped First Algorithm with a varying number of sensors and keeping all other values constant. All the readings in Table 5.1 are obtained after calculating the average of the values obtained from 25 test cases.

Table 5.1. Test Results for Most Overlapped First Algorithm Based on Coverage Gain

| TOTAL NUMBER OF SENSORS | Average Initial Coverage Area (IN %) | Average Final Coverage Area (IN %) | COVERAGE GAIN (IN %) | VARIANCE OF COVERAGE GAIN | STANDARD DEVIATION OF COVERAGE GAIN |
|---|---|---|---|---|---|
| 30 | 13.77 | 14.51 | 5.53 | 0.00134 | 0.03658 |
| 35 | 15.66 | 16.84 | 7.70 | 0.00233 | 0.04829 |
| 40 | 17.91 | 19.35 | 8.19 | 0.00180 | 0.04248 |
| 45 | 19.68 | 21.55 | 9.63 | 0.00134 | 0.03657 |
| 50 | 21.27 | 23.76 | 12.01 | 0.00329 | 0.05736 |
| 55 | 23.06 | 26.00 | 12.97 | 0.00256 | 0.05058 |
| 60 | 25.17 | 28.38 | 12.96 | 0.00197 | 0.04438 |
| 65 | 26.35 | 30.22 | 14.81 | 0.00196 | 0.04425 |
| 70 | 28.35 | 32.43 | 14.56 | 0.00179 | 0.04227 |
| 75 | 29.76 | 34.40 | 15.71 | 0.00157 | 0.03967 |
| 80 | 30.61 | 36.07 | 18.01 | 0.00258 | 0.05079 |
| 85 | 31.85 | 38.34 | 20.59 | 0.00270 | 0.05199 |

All through these tests, the number of static sensors are kept constant at 15 while the mobile sensors are varied from 15 to 70 sensors in increments of 5 for every 25 test cases. To obtain the data that are available in Table 5.1, 600 test cases were executed. Some values used in those test cases are as follows:

Number of Test Cases=25,

Total Area = 250000 pixels,

Number of Static Sensors = 15.

From Figure 5.1, we can see that the coverage gain is increasing with additional sensors. This finding shows that the algorithm is functioning effectively even when large numbers of sensors are used.

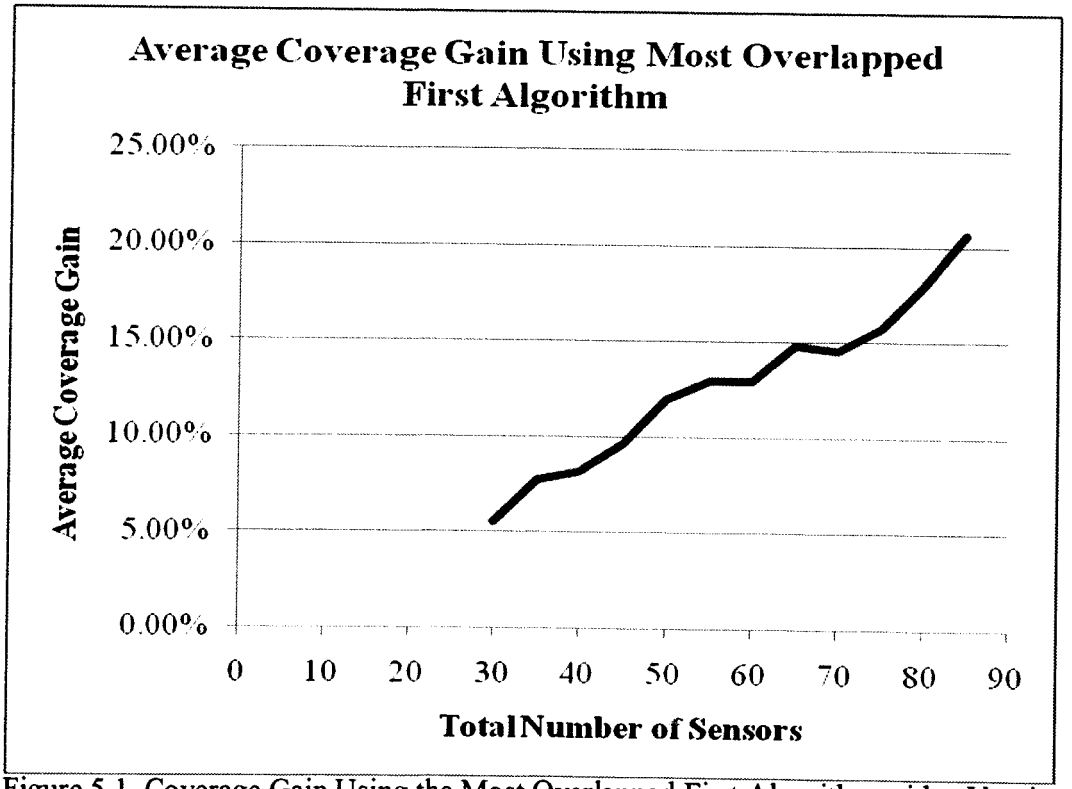


Figure 5.1. Coverage Gain Using the Most Overlapped First Algorithm with a Varying Number of Sensors

The coverage gain increased as the total number of sensors rose, keeping the covered area constant. The general increase in area covered is due to the fact that, as the number of sensors increased proportionately, the number of sensor overlaps also increased. Applying the Most Overlapped First Algorithm minimizes these overlaps; the net result is an increase in coverage gain. From the data obtained, the Average Coverage Gain, variance and standard deviation are calculated and displayed in Table 5.1, which assists in analyzing the effectiveness of the algorithm.

## 5.2. Test Case 2: Highest Coverage Gain Algorithm

Tests are performed using the Highest Coverage Gain Algorithm with a varying number of sensors and keeping all other values constant. All readings in Table 5.2 are obtained after calculating the average of the values obtained from 25 test cases. To obtain the data that are available in Table 5.2, 600 tests were executed. Some of the values used in those test cases are as follows:

Number of Test Cases=25,

Total Area = 250,000 pixels,

Number of Static Sensors = 15,

Radius of the Sensor = 20 pixels,

Maximum Number of Moves = 8,

Weight for Coverage Area (W1) = 0.5, and

Weight for Move Count (W2) = 0.5.

All these values remained constant for both algorithms and all the tests performed to collect the data. The idea behind keeping all the values constant and just varying the number of mobile sensors is to observe the change in coverage gain when the algorithms are applied.

Table 5.2. Test Results for Highest Coverage Gain Algorithm Based on Coverage Gain

| TOTAL NUMBER OF SENSORS | Average Initial Coverage Area (IN %) | Average Final Coverage Area (IN %) | COVERAGE GAIN (IN %) | VARIANCE OF COVERAGE GAIN | STANDARD DEVIATION OF COVERAGE GAIN |
|---|---|---|---|---|---|
| 30 | 13.83 | 14.58 | 5.55 | 0.00114 | 0.03370 |
| 35 | 15.70 | 16.91 | 7.94 | 0.00179 | 0.04233 |
| 40 | 17.61 | 19.21 | 9.31 | 0.00292 | 0.05400 |
| 45 | 19.75 | 21.68 | 9.88 | 0.00140 | 0.03748 |
| 50 | 21.56 | 23.98 | 11.56 | 0.00343 | 0.05859 |
| 55 | 23.15 | 26.14 | 13.15 | 0.00257 | 0.05071 |
| 60 | 24.98 | 28.19 | 12.98 | 0.00168 | 0.04095 |
| 65 | 26.04 | 30.32 | 16.59 | 0.00182 | 0.04272 |
| 70 | 27.77 | 32.38 | 16.86 | 0.00319 | 0.05646 |
| 75 | 29.35 | 34.34 | 17.14 | 0.00168 | 0.04100 |
| 80 | 30.45 | 35.99 | 18.35 | 0.00164 | 0.04051 |
| 85 | 31.90 | 38.39 | 20.45 | 0.00166 | 0.04077 |

Average coverage gain, variance and standard deviation of coverage gain are calculated using the test data that are collected. These values help us analyze the effectiveness of the algorithm.

From Figure 5.2, we can see that the coverage gain is increasing with additional sensors. This finding shows that the algorithm is functioning effectively even when a large numbers of sensors are used.

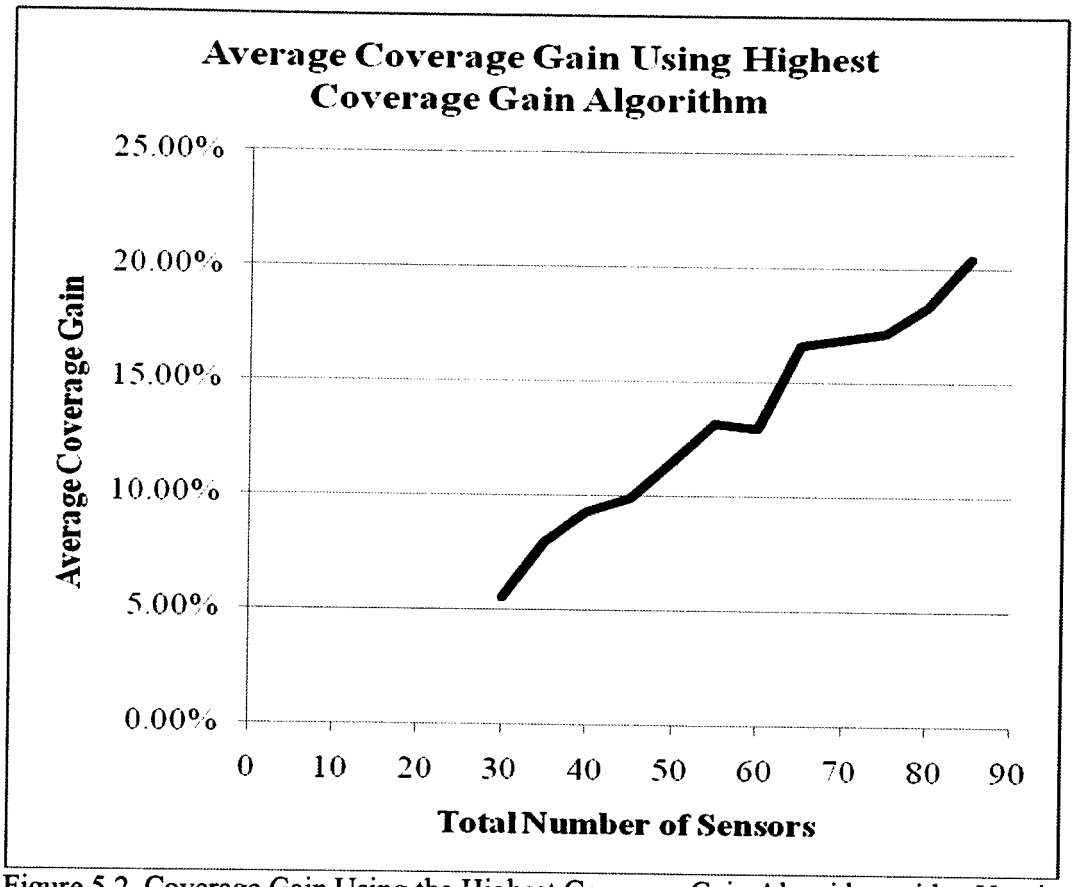


Figure 5.2. Coverage Gain Using the Highest Coverage Gain Algorithm with a Varying Number of Sensors

The readings obtained using the Highest Coverage Gain Algorithm are similar to the readings obtained using the Most Overlapped First Algorithm. If we compare the coverage gain percentage for the two algorithms in similar scenarios, we almost have similar values with a minor difference in the values of coverage gain obtained. The preceding two subsections are performed considering coverage gain as the metric. The next subsections compare the two algorithms based on coverage gain percentage as the metric with a varying number of sensors.

## 5.3. Comparison of Two Algorithms Based on Coverage Gain as the Metric

This subsection compares the two algorithms with coverage gain as the metric. We can see from the data in Table 5.3 that, when varying the number of sensors, both algorithms achieved a similar value for coverage gain. One reason for a high coverage gain value in both cases is due to the fact that more mobile sensors are used, keeping the static sensors constant. Large number of mobile sensors were used in test cases to observe the effectiveness of these algorithms when there are several moves in a sensor network.

Table 5.3. Comparison of Two Algorithms Based on Coverage Gain as the Metric

| TOTAL NUMBER OF SENSORS | COVERAGE GAIN USING MOST OVERLAPPED FIRST ALGORITHM (IN %) | COVERAGE GAIN USING HIGHEST COVERAGE GAIN ALGORITHM (IN %) |
|---|---|---|
| 30 | 5.53 | 5.55 |
| 35 | 7.70 | 7.94 |
| 40 | 8.19 | 9.31 |
| 45 | 9.63 | 9.88 |
| 50 | 12.01 | 11.56 |
| 55 | 12.97 | 13.15 |
| 60 | 12.96 | 12.98 |
| 65 | 14.81 | 16.59 |
| 70 | 14.56 | 16.86 |
| 75 | 15.71 | 17.14 |
| 80 | 18.01 | 18.35 |
| 85 | 20.59 | 20.45 |

From Figure 5.3, we can see that the coverage gain for any given number of sensors is almost equal for both algorithms. This finding shows that both algorithms perform in a similar manner to achieve the coverage gain for a given sensor network.

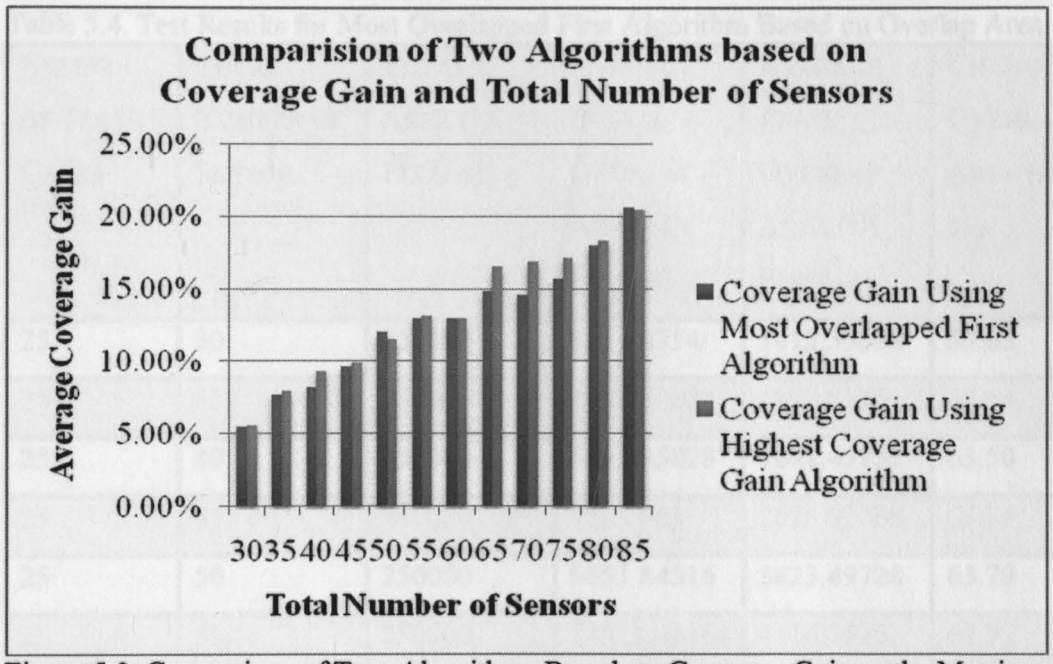


Figure 5.3. Comparison of Two Algorithms Based on Coverage Gain as the Metric

## 5.4. Test Case 3: Most Overlapped First Algorithm

The second metric that was considered to test the efficiency of the algorithm was overlap area. The smaller the overlap area in a sensor network, the more effective the overall sensor network is. By performing tests using a varying number of sensors, we calculated the percentage change in the overlap area. Altogether, 600 tests were conducted to minimize the error percentage in the readings gathered. Keeping the static sensors constant at 15, the mobile sensors were varied from 15 to 70 in increments of 5. All other data were kept constant through the experiments.

From Figure 5.3, we can see that the coverage gain for any given number of sensors is almost equal for both algorithms. This finding shows that both algorithms perform in a similar manner to achieve the coverage gain for a given sensor network.

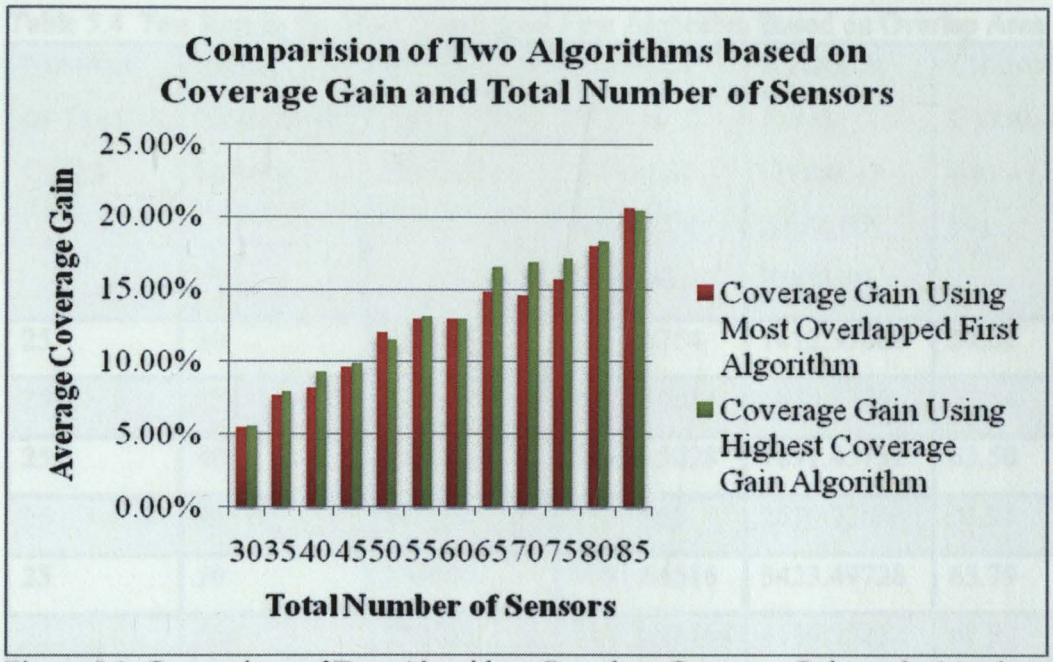


Figure 5.3. Comparison of Two Algorithms Based on Coverage Gain as the Metric

## 5.4. Test Case 3: Most Overlapped First Algorithm

The second metric that was considered to test the efficiency of the algorithm was overlap area. The smaller the overlap area in a sensor network, the more effective the overall sensor network is. By performing tests using a varying number of sensors, we calculated the percentage change in the overlap area. Altogether, 600 tests were conducted to minimize the error percentage in the readings gathered. Keeping the static sensors constant at 15, the mobile sensors were varied from 15 to 70 in increments of 5. All other data were kept constant through the experiments.

We can see from the data in Table 5.4 that, for a varying number of sensors, the Most Overlapped First Algorithm achieved a change in overlap area of at least 50% in each case.

Table 5.4. Test Results for Most Overlapped First Algorithm Based on Overlap Area

| Number of Test Cases | Total Number of Sensors | Total Area (in Pixels) | Average Initial Overlap Area (in Pixels) | Average Final Overlap Area (in Pixels) | Change in Overlap Area (in %) |
|---|---|---|---|---|---|
| 25 | 30 | 250000 | 3271.8754 | 1412.35668 | 54.05 |
| 25 | 35 | 250000 | 4823.30084 | 1879.3776 | 57.54 |
| 25 | 40 | 250000 | 5487.35828 | 1891.45732 | 63.50 |
| 25 | 45 | 250000 | 7353.953 | 2670.22068 | 63.75 |
| 25 | 50 | 250000 | 9651.84516 | 3423.49728 | 63.79 |
| 25 | 55 | 250000 | 11473.46164 | 4116.75252 | 62.99 |
| 25 | 60 | 250000 | 12473.30484 | 4436.30828 | 64.10 |
| 25 | 65 | 250000 | 15794.52728 | 6138.82752 | 60.85 |
| 25 | 70 | 250000 | 17101.22416 | 6898.82012 | 59.14 |
| 25 | 75 | 250000 | 19844.9698 | 8253.27456 | 58.39 |
| 25 | 80 | 250000 | 24001.71676 | 10356.19204 | 56.48 |
| 25 | 85 | 250000 | 27186.78896 | 10962.0546 | 59.57 |

Some of the data that are kept constant in both test cases are as follows:

Radius of the Sensor = 20 pixels,

Maximum Number of Moves = 8,

Weight for Coverage Area (W1) = 0.5, and

Weight for Move Count (W2) = 0.5.

From Figure 5.4, we can see that the percentage change in the overlap area for any given number of sensors is at least 50%. This finding shows that, for any given number of sensors, the algorithm effectively reduces the overlap area by at least half of the overlap area present before applying the algorithm for the given sensor networks.

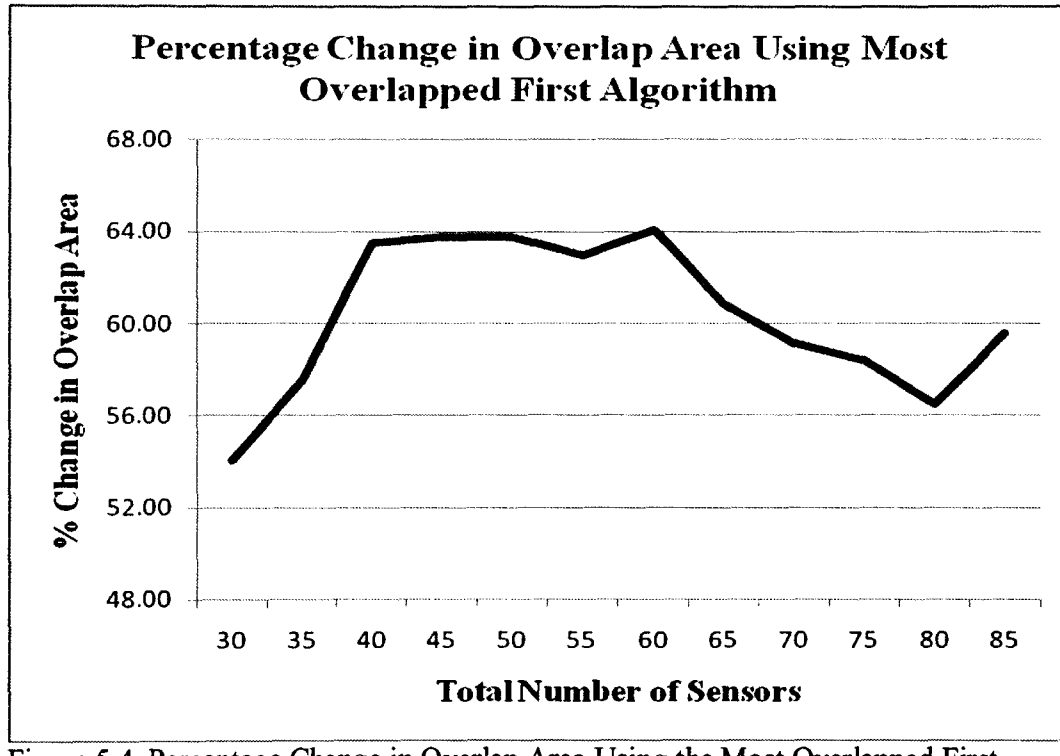


Figure 5.4. Percentage Change in Overlap Area Using the Most Overlapped First Algorithm

## 5.5. Test Case 4: Highest Coverage Gain Algorithm

The Highest Coverage Gain Algorithm is tested based on overlap area as the metric. The smaller the overlap area in a sensor network, the more effective the overall sensor network is. By performing tests using a varying number of sensors, we calculated the percentage change in overlap area. Altogether, 600 tests were conducted to minimize the error percentage in the readings gathered. Keeping the static sensors constant at 15,

mobile sensors were varied from 15 to 70 in increments of 5. All other data were kept constant through the experiments.

We can see from the data in Table 5.5 that, when varying the number of sensors, the Highest Coverage Gain Algorithm achieved a change in overlap area of at least 50% in each case.

Table 5.5. Test Results for Highest Coverage Gain Algorithm Based on Overlap Area

| NUMBER OF TEST CASES | TOTAL NUMBER OF SENSORS | TOTAL AREA (IN PIXELS) | AVERAGE INITIAL OVERLAP AREA (IN PIXELS) | AVERAGE FINAL OVERLAP AREA (IN PIXELS) | CHANGE IN OVERLAP AREA (IN %) |
|---|---|---|---|---|---|
| 25 | 30 | 250000 | 3132.29176 | 1255.06284 | 55.71 |
| 25 | 35 | 250000 | 4743.29168 | 1708.55716 | 58.10 |
| 25 | 40 | 250000 | 6233.11504 | 2232.32052 | 62.22 |
| 25 | 45 | 250000 | 7175.31768 | 2358.0208 | 66.15 |
| 25 | 50 | 250000 | 8938.92824 | 2870.15616 | 66.66 |
| 25 | 55 | 250000 | 11238.35284 | 3757.94808 | 66.17 |
| 25 | 60 | 250000 | 12943.00664 | 4931.57052 | 61.83 |
| 25 | 65 | 250000 | 16588.8018 | 5892.72264 | 64.49 |
| 25 | 70 | 250000 | 18542.62888 | 7018.14488 | 61.48 |
| 25 | 75 | 250000 | 20865.42676 | 8402.67608 | 59.95 |

This finding shows that, for any given number of sensors, the algorithm effectively reduces the overlap area by at least half of the overlap area present before applying the algorithm for a given sensor network. The important point that can be noted from the Table 5.5 readings is that the percentage change in overlap area is between the range of 55% and 66%, a number which is considerably high due to the fact that the ratio

of mobile sensors used is high when compared to static sensors that are kept constant at 15 for all tests.

From Figure 5.5, we can see that the percentage change in overlap area for any given number of sensors is at least 50%.

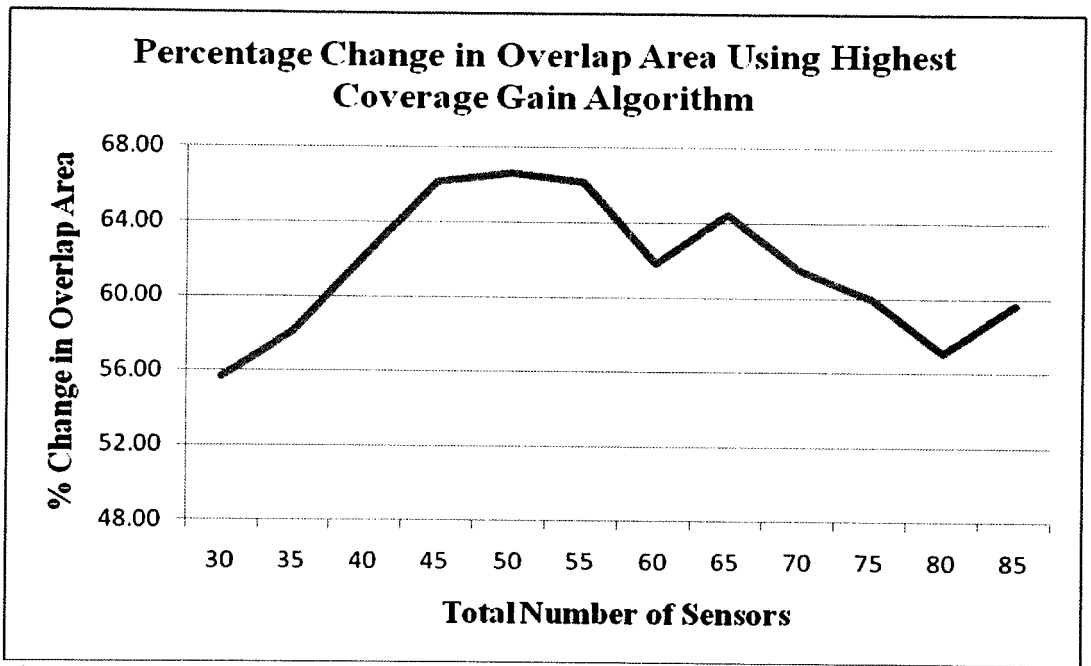


Figure 5.5. Percentage Change in Overlap Area Using the Highest Coverage Gain Algorithm

## 5.6. Comparison of Two Algorithms Based on Overlap Area as the Metric

This subsection compares the two algorithms based on the percentage change in overlap area with a varying number of sensors. This metric is important to consider because it shows how effective the algorithm is in reducing the overlap area.

We can see from the data in Table 5.6 that, when varying the number of sensors, both algorithms achieved a similar value for the percentage change in overlap area.

Another important observation that can be made from the data in Table 5.6 is that, for the given number of sensors, there is change in overlap area of at least 50%.

Table 5.6. Comparison of Two Algorithms Based on Overlap Area as the Metric

| TOTAL NUMBER OF SENSORS | PERCENTAGE CHANGE IN OVERLAP AREA USING MOST OVERLAPPED FIRST ALGORITHM | PERCENTAGE CHANGE IN OVERLAP AREA USING HIGHEST COVERAGE GAIN ALGORITHM |
|---|---|---|
| 30 | 54.05 | 55.71 |
| 35 | 57.54 | 58.10 |
| 40 | 63.50 | 62.22 |
| 45 | 63.75 | 66.15 |
| 50 | 63.79 | 66.66 |
| 55 | 62.99 | 66.17 |
| 60 | 64.10 | 61.83 |
| 65 | 60.85 | 64.49 |
| 70 | 59.14 | 61.48 |
| 75 | 58.39 | 59.95 |
| 80 | 56.48 | 57.06 |
| 85 | 59.57 | 59.68 |

When these values are plotted on a graph with the varying number of sensors on the horizontal axis and the percentage change in overlap area on the vertical axis, we will observe that the percentage change in overlap area does not fluctuate based on the total number of sensors. The constant change in overlap area irrespective of the number of sensors determines the effectiveness of the algorithms. This conclusion can be clearly understood from Figure 5.6 which plots the percentage change in overlap area for the two algorithms.

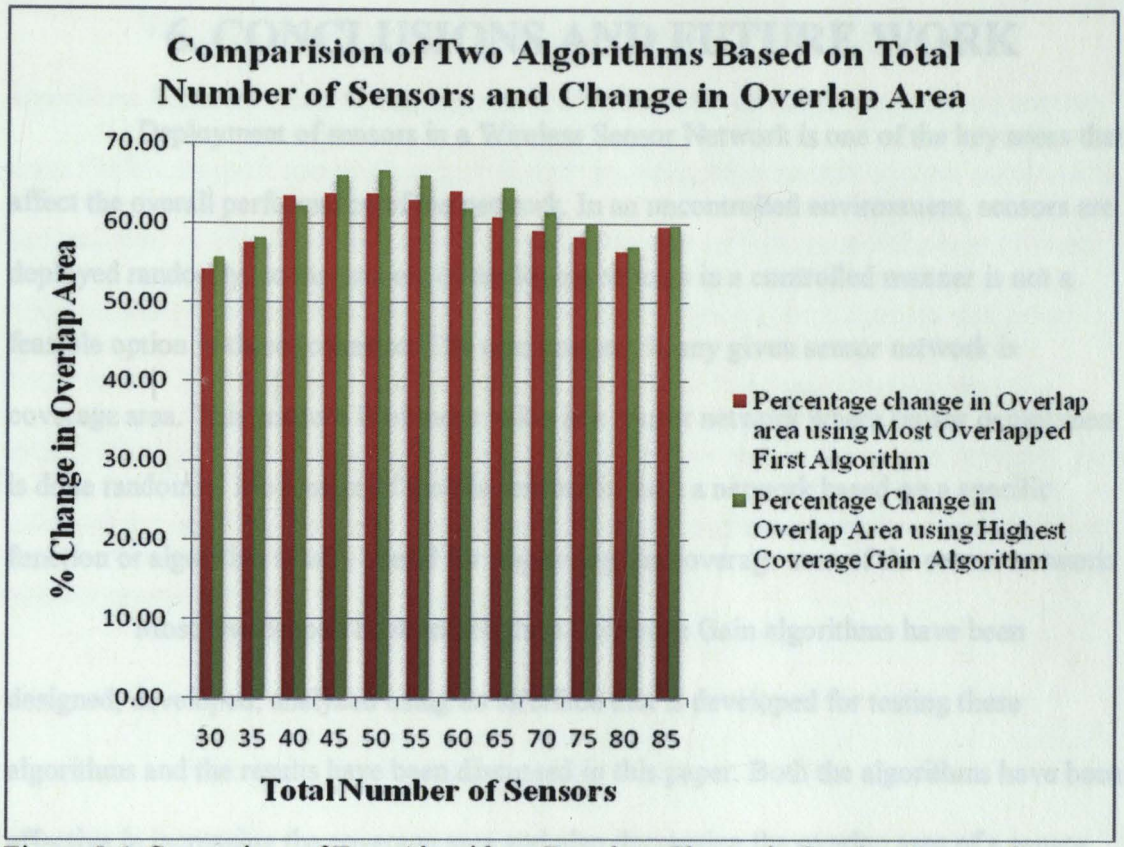


Figure 5.6. Comparison of Two Algorithms Based on Change in Overlap Area as the Metric

# 6. CONCLUSIONS AND FUTURE WORK

Deployment of sensors in a Wireless Sensor Network is one of the key areas that affect the overall performance of the network. In an uncontrolled environment, sensors are deployed randomly, so the process of deploying sensors in a controlled manner is not a feasible option in these scenarios. The next concern in any given sensor network is coverage area. This concern is of more value in a sensor network where sensor deployment is done randomly. Movement of mobile sensors in such a network based on a specific function or algorithm is very useful for improving the coverage area of the sensor network.

Most Overlapped First and Highest Coverage Gain algorithms have been designed, developed, analyzed using an interface that is developed for testing these algorithms and the results have been discussed in this paper. Both the algorithms have been effective in increasing the coverage area and also decreasing the overlap area of a sensor network. The algorithms also help us achieve efficient energy consumption by decreasing the number of moves for mobile sensors. The process of decreasing number of moves is done using the Utility Function evaluation that plays a major role in the two algorithms. With the results from experiments conducted on these two algorithms, we can conclude that both algorithms function effectively; applying these algorithms to a sensor network can help us achieve better coverage.

Some of the issues encountered while developing these algorithms is factors to be included in calculating a Utility Function. These algorithms can be implemented in real world sensor networks using a base station for communicating with mobile sensors regarding the movement that needs to be made in the network.

This paper described Most Overlapped First and Highest Coverage Gain algorithms based on two different metrics in a sensor network: coverage area and overlap area. Further research can be done on this topic by using other metrics such as move count, optimal number of sensors for a given network, the ratio of static to mobile sensors in an efficient sensor network and the variation of weight values for coverage area and move count in the Utility Function. Another area of research on this topic can be done by iteratively applying the algorithms on a sensor network to see the changes in coverage gain achieved for iteration. This also helps to analyze which algorithms works effectively when applied repetitively on a wireless sensor network

# REFERENCES


1. Ruiz, L., Nogueira, J., & Loureiro, A. (Feb 2003). MANNA: a management architecture for wireless sensor networks. *IEEE Communications Magazine , 41* (2), 116 - 125.

2. Ming Zhang Xiaojiang Du Nygard, K. (Oct 2005). Improving Coverage Performance in Sensor Networks by Using Mobile Sensors. *Military Communications Conference, 5*, pp. 3335 – 3341. Atlantic City, NJ.

3. Al-Karaki, J. K. (Dec 2004). Routing Techniques in Wireless Sensor Networks: a Survey. *IEEE Wireless Communications , 11* (6), 6-28.

4. Seyit A. Camtepe, B. Y. (Mar 2005). *Key Distribution Mechanisms for Wireless Sensor Networks: a Survey.* Technical Report, Rensselaer Polytechnic Institute, Computer Science Department, Troy, New York.

5. Xiang-Yang Li, P.-J. W. (2003). Coverage in Wireless Ad Hoc Sensor Networks. *IEEE Transactions on Computers , 52* (6), 753-763.

6. Mihaela Cardei,. J. (Feb 2006). Energy-efficient Coverage Problems in Wireless Ad-hoc Sensor Networks. *Computer Communications , 29* (4), 413-420.

7. K.E. Nygard, Faraz Katib (Mar 2008). Local Spiral Search in Sensor Coverage Problems, Dept. of Computer Science, North Dakota State University, unpublished notes.