DYNAMIC ALGORITHMS FOR SENSOR SCHEDULING AND ADVERSARY PATH PREDICTION

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

By

Shivendushital Pyarelal Pandey

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

Major Department:
Computer Science

June 2010
Fargo, North Dakota

North Dakota State University
Graduate School

Title

DYNAMIC ALGORITHMS FOR SENSOR SCHEDULING

AND ADVERSARY PATH PREDICTION

By

SHIVENDUSHITAL PYARELAL PANDEY

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

Pandey, Shivendushital Pyarelal, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, June 2010. Dynamic Algorithms for Sensor Scheduling and Adversary Path Prediction. Major Professor: Dr. Kendall Nygard.

In this thesis we describe three new dynamic, real time and robust sensor scheduling algorithms for intruder tracking and sensor scheduling. We call them Tactic Association Based Algorithm (**TABA**), Tactic Case Based Algorithm (**TCBA**) and Tactic Weight Based Algorithm (**TWBA**). The algorithms are encoded, illustrated visually, validated, and tested. The aim of the algorithms is to efficiently track an intruder or multiple intruders while minimizing energy usage in the sensor network by using real time event driven sensor scheduling. What makes these intrusion detection schemes different from other intrusion detection schemes in the literature is the use of historical data in path prediction and sensor scheduling.

The **TABA** uses sequence pattern mining to generate confidences of movement of an intruder from one location to another location in the sensor network. **TCBA** uses the Case-based reasoning approach to schedule sensors and track intruders in the wireless sensor network. **TWBA** uses weighted hexagonal representation of the sensor network to schedule sensors and track intruders.

In this research we also introduce a novel approach to generate probable intruder paths which are strong representatives of the paths intruders would take when moving through the sensor network.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Kendall Nygard for his continuous support, patience, motivation and guidance. I thank my parents, my family, Sumali and Eric iverson for their constant motivation to complete this thesis.

DEDICATION

I would like to dedicate this paper to my guru, Dr. Kendall Nygard.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CHARTS

1. INTRODUCTION

Wireless sensors are small programmable electronic devices that measure a physical phenomenon and transmit the acquired information to a base station. Wireless sensor networks consist of a large number of autonomous wireless sensor nodes. A wireless sensor network is a cohesive unit in which multiple sensors work together to achieve a common sensing goal. The successful performance of a sensor network depends on the successful operation of each sensor in the network. Wireless sensors have limited communication range, power supply, processing power and memory. Replacing a sensor in a sensor network is usually very difficult and expensive (1). Careful design and implementation is required for optimal and prolonged operation of the sensor network.

In order to fully realize the potential of a sensor network, energy awareness has to be incorporated into every stage of the network's design and operation (2). In a wireless sensor network, the three main areas of operation where maximum energy is consumed are communication, sensing and computing. The rate at which a sensor uses battery power is largest when communicating (3). Thus, optimizing the communication cost is essential for the prolonged operation of the sensor network. When the communication cost has been optimized, the power demands for sensing and computing operations should be considered. Thus, it is important to optimize the sensing and computing cost for prolonged functioning of the sensor network.

Wireless sensor networks have wide applications, such as, target area monitoring, environmental monitoring, inventory monitoring, intruder tracking etc. (4). Intruder tracking is one of the most important applications of wireless sensor networks. In a sensor network being used to track intruders, the two critical operations that the sensors perform are monitoring (sensing and computing) and reporting (communication). When monitoring (sensing/computing), sensor nodes are required to detect and track the movement of mobile target. When reporting (communication), the sensors sensing the target report their discoveries to the base station. These two operations are interleaved during the entire object tracking process. Most of the studied intruder tracking algorithms use dynamic duty scheduling to save energy and track intruder (5). The duty cycle of a sensor consists of work and sleep cycles. Dynamic sensor scheduling increases the network life time as fewer sensors are turned on at given time in a sensor network.

In an object tracking sensor network when the communication cost in the network has been optimized, we are presented with the problem of developing real time and energy efficient sensor scheduling and tracking algorithm to track multiple intruders in the sensor network. To be energy efficient and to be able to work real time, the tracking algorithm should enable dynamic and selective sensor scheduling. The algorithm should be able to adjust to the changing topology of the sensor network and the algorithm should be robust. This thesis concentrates on solving the problem creating an energy efficient algorithm with the characteristics mentioned above.

In this thesis we describe three new dynamic, real time and robust sensor scheduling algorithms for target tracking. We call them Tactic Association Based Algorithm (**TABA**), Tactic Case Based Algorithm (**TCBA**) and Tactic Weight Based Algorithm (**TWBA**). The algorithms are coded, illustrated visually, validated, and tested. The aim of the algorithms is to efficiently track an intruder or multiple intruders while minimizing energy usage in the sensor network by using real time, event driven sensor scheduling. What makes these intrusion detection schemes different from other intrusion detection schemes in the literature, is the use of historical data in path prediction and sensor scheduling.

The **TABA** uses sequence pattern mining to generate confidences of movement of an intruder from one location to another in the sensor network. The generated confidences form the basis for sensor scheduling and intruder tracking in the sensor network. **TCBA** uses the Case-based reasoning approach to schedule sensors and track intruders in the wireless sensor network. The **TCBA** compares the current intuder location and the number of steps required by the intruder to reach the current location with the paths in the historical paths database to find simmilar cases. The generated cases are used for scheduling the sensors and tracking the intruder in the sensor network. **TWBA** we use the hexagonal representation of the surveillance area. Each hexagon in the grid is assigned a weight. The weight represents the complexity of the terrain that the hexagon encompasses. The weight is used to generate movement probabilities from each hexagon in the grid. The movement probability includes the probability of movement of an intruder from the current hexagon to all the adjacent hexagons. The movement probabilities are the basis for scheduling in **TWBA**.

In this research we have also introduced a novel approach to generate probable intruder paths which are strong representatives of the paths intruder would take when moving through the sensor network. We use the concept of terrain complexity discussed above to generate probable intruder paths. The intruder is assumed to be an intelligent entity; he is aware of the terrain complexity around him and tries to find a path to a selected destination through a least complex terrain. The representative paths generated by using this method can be used for planning and designing of the sensor network and also in sensor scheduling.

2. LITERATURE REVIEW

2.1. Overview

Sensor networks have a wide range of application, one of the most important being intruder tracking. However, there are a number of problems that need to be addressed before deploying sensors for intruder tracking in real time applications. One of the most critical issues in intruder tracking is to save energy while providing meaningful tracking. Other issues are maintaining sufficient sensing coverage, network connectivity and fault tolerance. The most common approach used for saving energy and extending the lifetime of the sensor network, is to dynamically schedule duty cycles of the sensors (5). The duty cycle of a sensor consists of work and sleep cycles. When using dynamic sensor scheduling, fewer sensors are turned on at any given time and hence the life of the sensor network is effectively increased.

A number of scheduling schemes have been designed for hierarchical and non-hierarchical sensor networks. Each design uses a number of assumptions for power supply, network life time, network structure, sensor placement, detection model, sensing area, transmission range, failure model, sensor mobility and localization. In this chapter we will discuss research work related to intruder tracking algorithms and their design.

2.2. Background Study

One of the ways to maximize battery life of a sensor and hence increase network life time is to schedule the sensors sleep and work modes. In the work mode the sensor is either sensing or communicating. In the sleep mode all the processes of the sensor shut down except for the wakeup timer or a receiver. While in sleep mode the sensor consumes considerably less energy than it those in the work mode (6). Besides the most commonly used sensor modes discussed above, there are other sensor modes as well. In (7), the sensors work with three modes, monitor, observe and deep sleep. In the monitor mode both the sensing unit and the radio unit are turned ***ON***. In the observe mode, only the sensing unit is turned ON and in the deep sleep mode both the sensing unit and the receiver unit is turned OFF. Once in Deep Sleep mode the sensor uses a wakeup timer to switch to either the Monitor mode or Observe mode. Though the deep sleep mode saves considerable battery power, the transition between the modes take both energy and time. In many scenarios it might not be feasible to have extended wakeup time (8).

In (9), the authors use non-myopic sensor scheduling to increase network life time and preserve accuracy of target's position estimate. A hierarchical sensor network is used in the study. Sensors used in the sensor network are of two types, type A and type B. Type A sensors are used to collect measurements and type B sensors are used to collect, process and schedule measurements. To schedule a sequence of *n* sensing actions, the type B sensor holding the target belief computes the minimum energy sequence that satisfies the tracking accuracy constraint *n* steps in the future. Uniform cost search on an energy tree is used to implement scheduling.

In (10), the authors propose surveillance and tracking system called as VigilNet. The VigilNet tracking operation has six phases, initial activation, initial target detection, wakeup, group aggregation, end-to-end report and base processing. Initially the sensors in the sensor network are put in sentry state or non-sentry state. Sentry nodes are either part of communication infrastructure or sensing coverage infrastructure and the non-sentry nodes sleep. When the target is initially detected by a sentry node, the non-sentry nodes are awakened to provide group-based tracking. Once awakened, all nodes that detect the same target join the same logic group. Each group is represented by a leader which maintains the identity of the group as the target moves through the area. Group members periodically report the intruder position to the group leader. Leader reports detection to the base after the number of member reports exceeds a certain threshold.

In (11), the authors propose a distributed scheme to increase network life time while maintaining preserving sensing coverage. In this scheme, a sensor in the sensor network can turn ***OFF*** if its sensing area is completely covered by adjacent sensors. The scheme works in cycles and in each cycle the sensors obtain the adjacent sensor's location information. If the sensing area of the sensor is completely covered by its adjacent sensor, the sensor enters ***ready-to-off*** state, sets a random timer and declares its state to the adjacent sensors. In order to avoid multiple adjacent sensors turning ***OFF*** simultaneously and create a hole in the sensor network, a back-off strategy is used.

3. THE SURVEILLANCE AREA

3.1. Overview

The area being monitored is divided into mutually disjoint hexagonal cells. All the hexagonal cells are of equal size and each cell has a weight associated with it. The weight represents the probability that an intruder will move in to the hexagon. An example of the weight is the complexity of the terrain the hexagon encompasses. Another example of weight would be the knowledge of an intruder about the presence of a sensor in a given hexagon. In this study, the weight associated with a hexagon represents the complexity of the terrain the hexagon encompasses. The weight aka terrain complexity ranges from one to nine, wherein one represents the least complex terrain and nine represents the most complex terrain. Thus, if the terrain complexity of a hexagon is one, there is a large probability that the intruder would move into the hexagon and if the terrain complexity of a hexagon is nine, it is very less probable that an intruder will move into the hexagon. In this study the terrain complexity is randomly assigned to a hexagon.

3.2. Placement of Sensor in the Sensor Network

The sensors can be placed in the monitored area in a deterministic or a non-deterministic way. In either case, it is assumed that if a sensor falls in a hexagon it will provide complete sensing coverage to the entire area of the hexagon. If multiple sensors fall in a hexagon, only one of them will be active while others will sleep. There is no overlap in the coverage area of sensors. The hexagon which contains at least one sensor will be called a **Sensor hexagon** (**$\textbf{Sensor}_{\textbf{Hexagon}}$**). The hexagon in which no sensor is placed, will have no sensor coverage and will be called a **hole** (**$\textbf{Hole}_{\textbf{Hexagon}}$**).

In Figure 3.1. the hexagons shaded in light red are covered by a sensor and the hexagons shaded in green are holes. In this study, we can control the percentage of holes and sensors in the area being monitored. The distribution of sensors is non-deterministic.

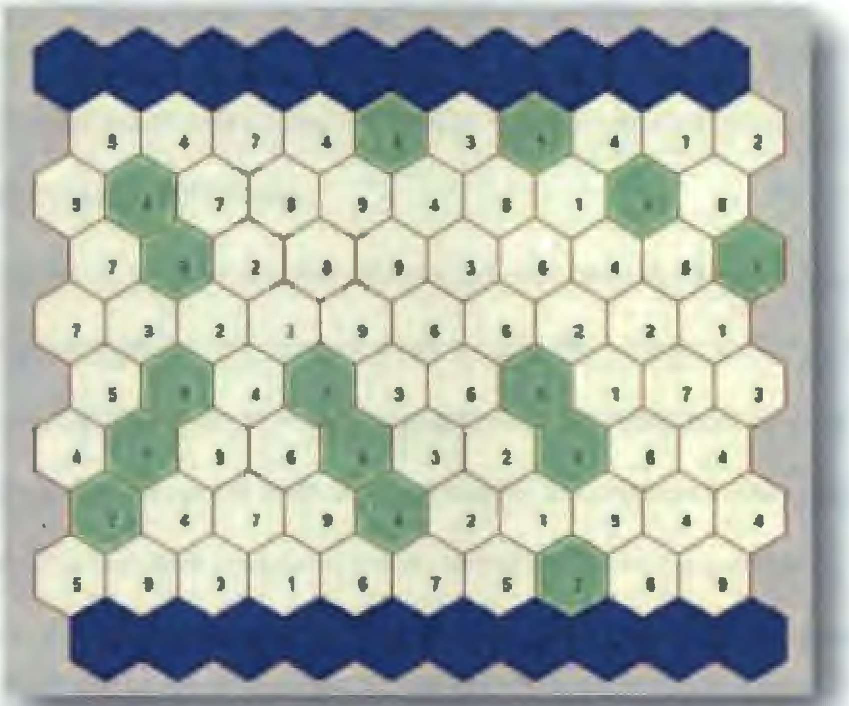


Figure 3.1. Hexagonal Representation of Surveillance Area

3.3. Sensors

The sensors used in the sensor network are homogeneous. All the sensors are static. Each sensor maintains the list of adjacent sensors and the remaining battery life of each adjacent sensor. The duty of the sensor is to detect intruder as the intruder moves into its sensing range and communicates the detection to the base station. The communication scheme used in this study is described, in detail, in chapter 9.

3.4. The Role of Base Station

The processing and decision making in the sensor network is centralized at the base station. The sensor network has a single base station located at the bottom right corner of the grid. The base station maintains a list of all available sensors and their respective positions in the sensor network. The base station collects data from the entire sensor network and uses this data to schedule sensors and track intruders. The base station broadcasts the scheduling information to the sensors in the network.

3.5. Defining Adjacent Sensors

For our study, we have assumed that the intrusion can only start at the upper boundary of the monitored area, proceed down through the network and end at the lower boundary. All the hexagons at the upper boundary and lower boundary contain sensors and are always in an ***ON*** state. An ideal case for the coverage in the sensor network would be that each hexagon in the sensor network contains a sensor i.e. all hexagons are sensor hexagons ($\textbf{Sensor}_{\textbf{hexagon}}$). In this case if $\textbf{Sensor}_{\textbf{Hexagon}}$ $\textbf{S}_{\textbf{i}}$ shares a boundary with another $\textbf{Sensor}_{\textbf{Hexagon}}$ $\textbf{S}_{\textbf{j}}$, then $\textbf{S}_{\textbf{i}}$ would be the adjacent of $\textbf{S}_{\textbf{j}}$ and vice versa. This definition of adjacent sensors will not apply if there are holes in the sensor network. In this case, we say two $\textbf{Sensors}_{\textbf{Hexagon}}$ are adjacent if it is possible for the intruder to move from of one of the two $\textbf{Sensor}_{\textbf{Hexagon}}$ to the other without passing through any other $\textbf{Sensor}_{\textbf{Hexagon}}$.

4. THE INTRUSION MODEL

4.1. Overview

The aim of the intruder is to move through the area being monitored covering least complex terrain. The intruder can enter the area being monitored at the upper boundary (**Boundary$_{\text{Upper}}$**) and exit at the lower boundary (**Boundary$_{\text{Lower}}$**). The intruder has a fixed line-of-sight (**L$_{\text{Sight}}$**). Line-of-sight is the maximum distance an intruder can see in any direction from his present location. The Line-of-sight is measured in the units of hexagon. When we say that the line-of-sight of an intruder is two hexagons we mean that the intruder cannot see beyond two hexagons in any direction. The line of sight of the intruder is constant throughout the intrusion process. In Figure 4.1. the intruder is represented in the hexagon with plus sign and the line of sight is represented by the light blue shaded hexagons. The line of sight the

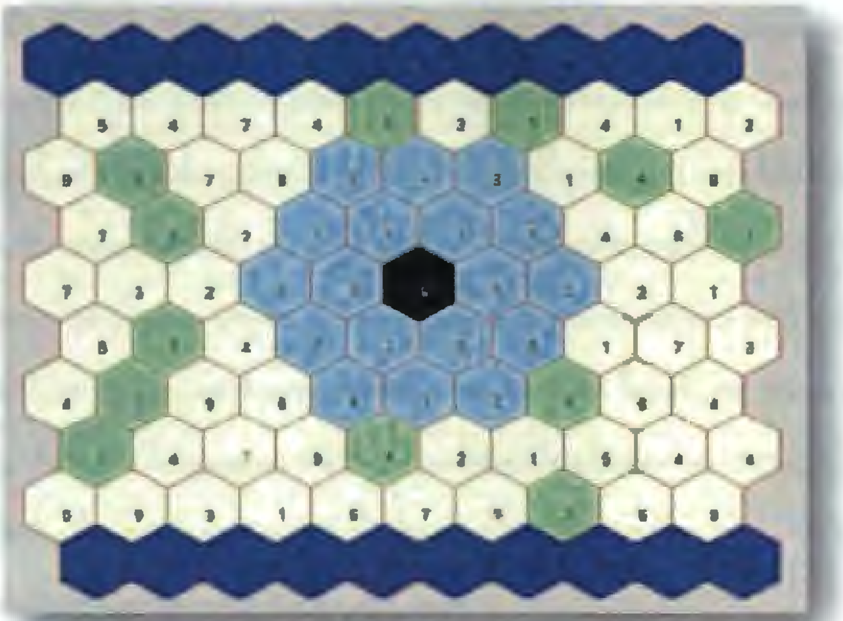


Figure 4.1. Line-of-Sight of an Intruder

intruder is two. The intruder can thus see only the hexagons highlighted by the light blue color.

The intruder is an intelligent entity. He is aware of the terrain complexity around him and tries to find his path to a selected destination through a least complex terrain. In a large sensor network, given the fact that the intruder has a limited line of sight, the intruder cannot decide the path that he is going to take from the upper boundary to the lower boundary in one go, but the intruder makes the decision of his movements in cycles. The sequence of intermediate cycles forms a complete path.

4.2. The Intrusion Algorithm

The intrusion algorithm uses variable $\boldsymbol{Intruder}_{\boldsymbol{IntermediatePath}}$ to store the sequence of intermediate steps an intruder takes during a single **decision cycle**. The variable $\boldsymbol{Intruder}_{\boldsymbol{Path}}$ is used to store the complete path of the intruder through the sensor network. The intruder randomly chooses the point of entry into the area being monitored. The point of entry is a hexagon on the upper boundary ($\boldsymbol{Boundary}_{\boldsymbol{Upper}}$). On reaching the point of entry, looking towards the lower boundary he randomly chooses a hexagon on the intermediate boundary, $\boldsymbol{Intermediate}_{\boldsymbol{Boundary}}$. Intermediate boundary is a set of hexagon that lies at the edge of the line-of-sight, looking towards the lower boundary. The intruder then takes the minimum terrain complexity path to the intermediate position on the intermediate boundary. He repeats the above process until he reaches a hexagon on the lower boundary. Algorithm uses **Dijkstra** algorithm to calculate the least complex path to the intermediate position. Table 4.1. lists all the abbreviations used in the intrusion algorithm.

Table 4.1. Abbreviations Used in the Pseudo Code for Intrusion Algorithm

Abbreviation	Meaning
L_{Sight}	Line of sight. The farthest the intruder can see from the present location
H_{Sight}	All the hexagons the intruder can see from the current location
$Intremediate_{Boundary}$	Set of hexagon that lies at the edge of the line-of-sight, looking towards the lower boundary
$Intruder_{Path}$	The path taken by the intruder in the sensor network
$Intruder_{IntermediatePath}$	Subset of $Intruder_{Path}$
$Intruder_{CurrentLocation}$	The current location of the intruder
$Intruder_{IntermediateLocation}$	Intermediate location of the intruder
Random(SET)	Function returns a random element from the set of elements
$Boundry_{Upper}$	The boundary of the sensor network from where the intrusion starts
$Boundry_{Lower}$	The boundary of the sensor network where the intrusion ends
APPEND($Intruder_{Path,}$ $Intruder_{IntermediatePath}$)	Function appends the element of $Intruder_{IntermediatePath}$ at the end of $Intruder_{Path}$

4.3. Pseudo Code: The Intrusion Algorithm

INPUT: Current Location of the intruder ($Intruder_{CurrentLocation}$)

Line of sight of intruder (L_{sight})

```
BEGIN
{
  $Intruder_{Path} = \Phi$
  $Intruder_{IntermediatePath} = \Phi$

    // randomly select the entry point
    $Intruder_{CurrentLocation} = Random(Boundary_{Upper})$

  // If the present position not on Lower boundary repeat
   WHILE($Intruder_{CurrentLocation} \notin Boundary_{Lower}$)
   {
     // randomly choose a hexagon at the intermediate position
     $Intruder_{IntermediateLocation} = Random(Intremediate_{Boundary})$

     // Take the shortest path to the intermediate position
     $Intruder_{IntermediatePath} = Dijkstra(Intruder_{IntermediateLocation}, H_{Sight})$

    APPEND($Intruder_{Path}$, $Intruder_{IntermediatePath}$)

    $Intruder_{CurrentLocation} = Intruder_{IntermediateLocation}$
  }
  Return $Intruder_{Path}$
}
```

5. USING SEQUENCE PATTERN MINING IN SENSOR SCHEDULING AND INTRUDER TRACKING

5.1. Overview

The Tactic Association Based Algorithm uses sequence pattern mining to schedule sensors and track intruders in a sensor network. The sequence pattern mining technique is used to derive association rules between two locations (Hexagons) in the surveillance area. Association rules define the support of a given hexagon in the sensor network and the confidence that an intruder will move from one location to the other location. Below we explain in detail the term associations, support and confidence and also present the algorithm used to generate them.

Association rules are generated from the historical paths. An association rule is an expression of form $\mathbf{S_i} \rightarrow \mathbf{S_j}$, where $\mathbf{S_i}$ and $\mathbf{S_j}$ hexagons. The intuitive meaning of such a rule is, the intruder paths which contain S_i tend to contain $\mathbf{S_j}$. An example of such a rule might be that "30% of intruders that pass through sensor node $\mathbf{S_i}$ also pass through sensor node $\mathbf{S_j}$".

Each association rule has two measures of value, support, and confidence. Support indicates the frequencies of the occurring patterns, and confidence denotes the strength of implication in the rule. The support of sensor hexagon $\mathbf{S_i}$ in the sensor network is the total number of path in the collection of historical paths that contain $\mathbf{S_i}$. The sensor node $\mathbf{S_i}$ has support, **s**, in the set of historical paths **P** if **s%** of historical paths in **P** contain $\mathbf{S_i}$; we denote **s = support ($\mathbf{S_i}$)**. The confidence, **c**, of the rule $\mathbf{S_i} \rightarrow \mathbf{S_j}$ in the set of historical paths **P** means **c%** of Path-Traversed in **P** that contain **Si** also contain **Sj**, which can be written as **(support ($\mathbf{S_i \cap S_j}$) / support ($\mathbf{S_i}$))** .

5.2. Algorithm

Table 5.1. lists all the abbreviations used in the algorithm below. For a sensor network the support value of each sensor in the sensor network is calculated. For each tuple ***(S_i, S_j)*** where **S_i** and **S_j** are adjacent sensors, confidence value is calculated and stored in the database.

Step 1: Figure 5.1. shows the pseudo code for step 1. For each sensor hexagon present in the sensor network, traverse every path present in the collection of historical paths. If the sensor hexagon is an element of a path, increase the support of the sensor hexagon by one. The support value obtained for the sensor hexagon after

```
BEGIN
{
  FOREACH $Sensor_{Hexagon}$ $S_i$
  {
    FOREACH Path $\in$ $Collection_{HistoricalPath}$
    {
      IF($S_i \in$ Path)
      {
        Support($S_i$) = Support($S_i$) +1
      }
    }
```

Figure 5.1. Using Sequence Pattern Mining Algorithm for Sensor Scheduling and Intruder Tracking: STEP 1

all the paths in the collection of historical paths has been traversed, is the final support of the sensor hexagon in the sensor network.

Step 2: Figure 5.2. shows the pseudo code for step 2. For two adjacent sensor hexagon **S_i** and **S_j** present in the sensor grid traverse every path present in the collection of historical paths. If the sensor hexagon **S_i** and **S_j** are the elements of a path and the intruder has moved to the hexagon **S_j** from **S_i**, increase the support of the expression **$S_i \rightarrow S_j$** by one. The support value obtained for the expression **$S_i \rightarrow S_j$** after all the paths in the collection of historical paths have been traversed is the final support of the expression $\mathbf{S_i} \rightarrow \mathbf{S_j}$ in the sensor network.

```
FOREACH $Sensor_{Hexagon}$ $S_i$
{
  FOREACH $Sensor_{Hexagon}$ $S_j$
  {
    IF($S_j \in Adjacent(S_i)$)
    {
      FOREACH Path $\in$ $Collection_{HistoricalPath}$
      {
        IF($S_i \in$ Path AND $S_j \in$ Path AND $Next(S_i, S_j)$)
        {
        $Support(S_i, S_j) = Support(S_i, S_j) + 1$
        }
      }
    }
  }
}
```

Figure 5.2. Using Sequence Pattern Mining Algorithm for Sensor Scheduling and Intruder Tracking: STEP 2

Step 3: Figure 5.3. shows the pseudo code for step 3. For two adjacent sensor hexagon $\mathbf{S_i}$ and $\mathbf{S_j}$ present in the sensor grid, the confidence of the intruder moving

```
FOREACH $Sensor_{Hexagon}$ $S_i$
 {
  FOREACH $Sensor_{Hexagon}$ $S_j$
  {
    IF($S_j \in Adjacent(S_i)$)
    {
     $Confidance(S_i, S_j) = Support(S_i, S_j) / Support(S_i)$
    }
  }
 }
}
```

Figure 5.3. Using Sequence Pattern Mining Algorithm for Sensor Scheduling and Intruder Tracking: STEP 3

from hexagon $\mathbf{S_i}$ to $\mathbf{S_j}$ is obtained by dividing the support of expression $\mathbf{S_i} \rightarrow \mathbf{S_j}$ by the support of $\mathbf{S_i}$.

Table 5.1. Abbreviations Used in the Sequence Pattern Mining Algorithm for Sensor Scheduling and Intruder Tracking

Abbreviation	**Meaning**
$\text{Collection}_{\text{HistoricalPath}}$	Set of historical intruder paths
Support(S_i)	The function calculates support of a sensor hexagon in the sensor network
Support(S_i, S_j)	The function calculates the support of the implication $\mathbf{S_i} \rightarrow \mathbf{S_j}$ in the sensor network
Next(S_i, S_j)	The function returns a true value if the intruder has moved directly from sensor hexagon S_i to sensor hexagon S_j, else the function returns a false value.
Confidance(S_i, S_j)	The function calculates confidence of intruder moving from sensor hexagon S_i to sensor hexagon S_j.

5.3. Pseudo Code: Sequence Pattern Mining Algorithm for Sensor Scheduling and Intruder Tracking

```
INPUT: Sensor grid ($\text{Sensor}_{\text{Grid}}$)
       Collection of historical paths ($\text{Collection}_{\text{HistoricalPath}}$)

1 BEGIN
2 {
```

```
FOREACH $\text{Sensor}_{\text{Hexagon}}\ S_i$
{
  FOREACH Path $\in \text{Collection}_{\text{HistoricalPath}}$
  {
    IF($S_i \in$ Path)
    {
      $\text{Support}(S_i) = \text{Support}(S_i) + 1$
    }
  }
}

FOREACH $\text{Sensor}_{\text{Hexagon}}\ S_i$
{
  FOREACH $\text{Sensor}_{\text{Hexagon}}\ S_j$
  {
    IF($S_j \in \text{Adjacent}(S_i)$)
    {
      FOREACH Path $\in \text{Collection}_{\text{HistoricalPath}}$
      {
        IF($S_i \in$ Path AND $S_j \in$ Path AND $\text{Next}(S_i, S_j)$)
        {
          $\text{Support}(S_i, S_j) = \text{Support}(S_i, S_j) + 1$
        }
      }
```

```
            }
        }
    }
  FOREACH $\text{Sensor}_{\text{Hexagon}}$ $S_i$
    {
     FOREACH $\text{Sensor}_{\text{Hexagon}}$ $S_j$
      {
        IF($S_j \in$ Adjacent($S_i$))
        {
          Confidence($S_i$, $S_j$) = Support($S_i$, $S_j$) / Support($S_i$)
        }
    }
  }
}
```

6. THE TACTIC ASSOCIATION BASED ALGORITHM

6.1. Overview

The **TABA** uses confidences generated using sequence pattern mining technique discussed in the previous section as a basis for scheduling and tracking intruders in the sensor network. The objective of the **TABA** algorithm is to provide effective intruder tracking while increasing the sensor network's life time.

As input, **TABA** needs the information about the present location of the intruder (***$Intruder_{CurrentLocation}$***), the confidence matrix which contains the confidence measures generated using sequence pattern mining technique and the minimum threshold value (***$Minimum_{Threshold}$***). Given the present location (***S_i***) of the intruder, adjacent sensor nodes (***S_j***) will be turned ON only when ***Confidence (S_i, S_j)*** is greater than the ***$Minimum_{Threshold}$***. The value of minimum threshold ranges from **0** to **100**. The **TABA** maintains a list (***$ListSensors_{ON}$***) of sensor which has to be turned ***ON***. The list is created real time at each intruder step. The function ***Adjacent ($Intruder_{CurrentLocation}$)*** returns a list of all sensors hexagons adjacent to the current intruder location. The function ***Confidence ($Intruder_{CurrentLocation}$, $Sensor_{Hexagon}$)*** returns the confidence of intruder moving from current intruder location to the given sensor hexagon. The function ***ADD ($ListSensor_{ON}$, $Sensor_{Hexagon}$)*** adds the given sensor hexagon to the list ***$ListSensor_{ON}$***.

6.2. Algorithm

Table 6.1. lists all the abbreviations used in the algorithm below. TABA works in three steps. Each step on the TABA algorithm is explained in the detail below.

Step 1: Figure 6.1. shows the pseudo code for step 1. The intruder is first detected on the upper boundary (***$Upper_{Boundary}$***). The TCBA is initialized on the first detection. The current intruder location (***$Intruder_{CurrentLocation}$***) and the current intruder step (***$Intruder_{CurrentStep}$***) are passed to the algorithm. The variable ***$ListSensors_{ON}$*** stores the ID of all the sensors ON in the previous step. **TABA** then checks to see if ***$ListSensors_{ON}$*** is empty. If ***$ListSensors_{ON}$*** is not empty, all the ***$Sensor_{Hexagon}$*** listed in ***$ListSensors_{ON}$*** are turned off and the ***$ListSensors_{ON}$*** is cleared.

```
BEGIN
{
  $ListSensors_{ON} = \Phi$
  WHILE($Intruder_{CurrentLocation} \notin Boundary_{Lower}$)
  {
     IF($ListSensors_{ON} \neq \Phi$)
     {
        FOREACH $Sensor_{Hexagon} \in ListSensors_{ON}$
        TURN OFF $Sensor_{Hexagon}$
     }
     $ListSensors_{ON} = \Phi$
```

Figure 6.1. The Tactic Association Based Algorithm: STEP 1

Step 2: Figure 6.2. shows the pseudo code for step 2. The algorithm compares the confidence of the intruder moving from the current intruder location to each

```
     FOREACH $Sensor_{Hexagon} \in Adjacent(Intruder_{CurrentLocation})$
     {
        IF( $Confidance(Intruder_{CurrentLocation}, Sensor_{Hexagon}) > Minimum_{Threshold}$)
        {
           ADD($ListSensor_{ON}, Sensor_{Hexagon}$)
           TURN ON $Sensor_{Hexagon}$
        }
```

Figure 6.2. The Tactic Association Based Algorithm: STEP 2

adjacent sensor hexagon with the minimum threshold. If the confidence is greater than the minimum threshold, the sensor is added to the ***ListSensor$_{ON}$*** and turned ON.

Step 3: Figure 6.3. shows the pseudo code for step 3. In the next step the intruder will move from his current location to a new location. If the new intruder location is a sensor hexagon and the sensor hexagon was turned ON in STEP 2, the

```
        IF( $Intruder_{NewLocation} \in SensorListON$)
        {
            $Intruder_{CurrentLocation} = Intruder_{NewLocation}$
        }
        ELSE
        {
            $New_{IntruderLocation} = FindIntruder(Intruder_{CurrentLocation})$
            $Intruder_{CurrentLocation} = New_{IntruderLocation}$
        }
    }
}
```

Figure 6.3. The Tactic Association Based Algorithm: STEP 3

intruder has been successfully tracked. If the intruder has been sucessfully tracked, STEP 1 through STEP 3 is repeated with the new intruder location. If the intruder has not been sucessfully tracked, that is, the intruder has moved to a sensor hexagon which was not turned on in the STEP 2, the recovery algorithm to trace the location of intruder is initialized. The new location of intruder is traced and and STEP 1 through STEP 3 is repeated with the new intruder location. The algorithm is terminated if the intruder moves to a location on the lower boundary.

Table 6.1. Abbreviations Used in the Tactic Association Based Algorithm

Abbreviation	**Meaning**
$Intruder_{CurrentLocation}$	Current intruder location
$Intruder_{NewLocation}$	New intruder location
$ListSensors_{ON}$	Set of probable locations ($Sensor_{Hexagon}$) where the intruder can move in the next step
$Collection_{HistoricalPaths}$	Collection of historical paths
Adjacent (S_i)	{ $Sensor_{Hexagon}$ | $\forall$ $Sensor_{Hexagon}$ are adjacent to $Sensor_{Hexagon}$ S_i}
Confidence($Intruder_{CurrentLocation}$, $Sensor_{Hexagon}$)	Function returns the confidence of intruder moving from current intruder location to the given sensor hexagon
ADD($ListSensor_{ON}$, $Sensor_{Hexagon}$)	Function adds the given sensor hexagon to list $ListSensor_{ON}$.

6.3. Pseudo Code: The Tactic Association Based Algorithm

INPUT: Confidence Vector

Current Location of the intruder ($Intruder_{CurrentLocation}$)

Minimum Threshold ($Minimum_{Threshold}$)

```
1 BEGIN
```

```
{
ListSensors$_{ON}$ = $\Phi$
WHILE(Intruder$_{CurrentLocation}$ $\notin$ Boundary$_{Lower}$)
{
   IF(ListSensors$_{ON}$ $\neq$ $\Phi$)
   {
      FOREACH Sensor$_{Hexagon}$ $\in$ ListSensors$_{ON}$
      TURN OFF Sensor$_{Hexagon}$
    }
    ListSensors$_{ON}$ = $\Phi$
    FOREACH Sensor$_{Hexagon}$ $\in$ Adjacent(Intruder$_{CurrentLocation}$)
    {
       IF( Confidence(Intruder$_{CurrentLocation}$,Sensor$_{Hexagon}$) > Minimum$_{Threshold}$)
       {
          ADD(ListSensor$_{ON}$, Sensor$_{Hexagon}$)
          TURN ON Sensor$_{Hexagon}$
       }
       IF( Intruder$_{NewLocation}$ $\in$ SensorListON)
      {
          Intruder$_{CurrentLocation}$ = Intruder$_{NewLocation}$
       }
       ELSE
       {
            $New_{IntruderLocation}$ = FindIntruder($Intruder_{CurrentLocation}$)
            $Intruder_{CurrentLocation}$ = $New_{IntruderLocation}$
        }
    }
  }
}
```

7. THE TACTIC CASE BASED ALGORITHM

7.1. Overview

Tactic Case Based Algorithm Algorithm uses the Case-based reasoning approach to schedule sensors and track intruders in the wireless sensor network. The current intruder location and current intruder steps are together used to predict the next probable location of the intruder in the wireless sensor network. As discussed in chapter **Error! Reference source not found.**, a historical path contains a list of sensor hexagons. The position of a sensor hexagon in the list represents the step at which the intruder reached the sensor hexagon. The **TCBA** compares the current intuder location and the current intruder step with the paths in the collection of historical paths to find simmilar cases. If **TCBA** finds a historical path which contains the current intruder location at current intruder step, the sensor hexagon present at next step in the historical path is added into the list of solutions. The objective of the **TCBA** is to provide effective intruder tracking while increasing the sensor network life time.

As input, **TCBA** needs information about the present location of the intruder ($\boldsymbol{Intruder_{CurrentLocation}}$), present intruder step ($\boldsymbol{Intruder_{CurrentStep}}$) and collection of historical intruder paths ($\boldsymbol{Collection_{HistoricalPaths}}$). **TCBA** maintains a list ($\boldsymbol{ListSensors_{ON}}$) of probable locations ($\boldsymbol{Sensor_{Hexagon}}$) the intruder would move in the next step. The function $\boldsymbol{Adjacent(Intruder_{CurrentLocation})}$ returns a set of all $\boldsymbol{Sensor_{Hexagon}}$ adjacent to the current intruder location. The function $\boldsymbol{EvaluatePath(Path, Intruder_{CurrentStep})}$ returns the $\boldsymbol{Sensor_{Hexagon}}$ at the current intruder step in the given historical path. Initially $\boldsymbol{ListSensors_{ON}}$ is empty. Table 7.1. lists all the abbreviations used in the algorithm below.

7.2. Algorithm

Step 1: Figure 7.1. shows the pseudo code for step 1. The intruder is first detected on the upper boundary (***$Upper_{Boundary}$***). The **TCBA** is initialized on the first detection. The current intruder location (***$Intruder_{CurrentLocation}$***) and the current intruder step (***$Intruder_{CurrentStep}$***) are passed to the algorithm. **TCBA** then checks to see if ***$ListSensors_{ON}$*** is empty. If ***$ListSensors_{ON}$*** is not empty, all the *$Sensor_{Hexagon}$* listed in ***$ListSensors_{ON}$*** are turned OFF and the ***$ListSensors_{ON}$*** is cleared.

```
BEGIN
{
 $ListSensors_{ON} = \Phi$
 WHILE($Intruder_{CurrentLocation} \notin Boundary_{Lower}$)
 {
    IF($ListSensors_{ON} \neq \Phi$)
    {
       FOREACH $Sensor_{Hexagon} \in ListSensors_{ON}$
       TURN OFF $Sensor_{Hexagon}$
     }
    $ListSensors_{ON} = \Phi$
```

Figure 7.1. The Tactic Case Based Algorithm: STEP 1

Step 2: Figure 7.2. shows the pseudo code for step 2. Each path in the collection of historical paths is evaluated to find those paths that contain

```
    FOREACH Path $\in Collection_{HistoricalPaths}$
    {
      $Sensor_{Hexagon}$ = EvaluatePath(Path, $Intruder_{CurrentStep}$)
      IF($Intruder_{CurrentLocation}$ == $Sensor_{Hexagon}$)
      {
          $Sensor_{Hexagon}$ = EvaluatePath(Path, $Intruder_{CurrentStep} + 1$)
        ADD($ListSensor_{ON}$, $Sensor_{Hexagon}$)
        TURN ON $Sensor_{Hexagon}$
      }
```

Figure 7.2. The Tactic Case Based Algorithm: STEP 2

Intruder$_{CurrentLocation}$ at ***Intruder***$_{CurrentStep}$ in the path. If such a path is found, the ***Sensor***$_{Hexagon}$ present at (***Intruder***$_{CurrentStep}$***+1***) is turned ON and added in the list of solutions (ListSensors$_{ON}$).

Step 3: Figure 7.3. shows the pseudo code for step 3. In the next step the intruder will move from his current location to a new location. If the new intruder location is a sensor hexagon and the sensor hexagon was turned on in STEP 2, the intruder has been successfully tracked. If the intruder has been sucessfully tracked, STEP 1 through STEP 3 is repeated with the new intruder location and new intruder step (***Previous Intruder Step + 1***). If the intruder has not been sucessfully tracked, that

```
        IF( $Intruder_{NewLocation} \in$ SensorListON)
        {
            $Intruder_{CurrentLocation}$ = $Intruder_{NewLocation}$
        {
        ELSE
         {
             $Intruder_{NewLocation}$ = FindIntruder($Intruder_{CurrentLocation}$)
             $Intruder_{CurrentLocation}$ = $New_{IntruderLocation}$
            }
        }
    }
}
```

Figure 7.3. The Tactic Case Based Algorithm: STEP 3

is, the intruder has moved to a sensor hexagon which was not turned on in STEP 2, the recovery algorithm to trace the location of intruder is initialized. The new location of intruder is traced and STEP 1 through STEP 3 is repeated with new intruder location location and new intruder step (***Previous Intruder Step + 1***). The algorithm is terminated if the intruder moves to a location on the lower boundary.

Table 7.1. Abbreviations Used in the Tactic Case Based Algorithm

Abbreviation	**Meaning**
$\text{Intruder}_{\text{CurrentLocation}}$	Current location of the intruder
$\text{Intruder}_{\text{NewLocation}}$	New intruder location
$\text{ListSensors}_{\text{ON}}$	Set of probable locations ($\text{Sensor}_{\text{Hexagon}}$) where the intruder can move in the next step
$\text{Collection}_{\text{HistoricalPaths}}$	Collection of historical paths
Adjacent (S_i)	$\{ \text{Sensor}_{\text{Hexagon}} \mid \forall \text{Sensor}_{\text{Hexagon}}$ are adjacent to $\text{Sensor}_{\text{Hexagon}}\ S_i \}$
EvaluatePath(Historical path, Intruder step)	The function returns the $\text{Sensor}_{\text{Hexagon}}$ at the given intruder step
$\text{Intruder}_{\text{CurrentStep}}$	Each movement of the intruder from one $\text{Sensor}_{\text{Hexagon}}$ to another $\text{Sensor}_{\text{Hexagon}}$ is a step of intruder. The number of steps taken by the intruder to reach the current intruder location is current intruder step ($\text{Intruder}_{\text{CurrentStep}}$).

7.3. Pseudo Code: The Tactic Case Based Algorithm

INPUT: Current location of intruder ($\text{Intruder}_{\text{CurrentLocation}}$)

```
        Current intruder step ($Intruder_{CurrentStep}$)
        Collection of historical paths ($Collection_{HistoricalPaths}$)
BEGIN
{
  $ListSensors_{ON} = \Phi$
  WHILE($Intruder_{CurrentLocation} \notin Boundary_{Lower}$)
  {
    IF($ListSensors_{ON} \neq \Phi$)
    {
      FOREACH $Sensor_{Hexagon} \in ListSensors_{ON}$
      TURN OFF $Sensor_{Hexagon}$
    }
    $ListSensors_{ON} = \Phi$
    FOREACH Path $\in Collection_{HistoricalPaths}$
    {
      $Sensor_{Hexagon}$ = EvaluatePath(Path, $Intruder_{CurrentStep}$)
      IF($Intruder_{CurrentLocation} == Sensor_{Hexagon}$)
      {
          $Sensor_{Hexagon}$ = EvaluatePath(Path, $Intruder_{CurrentStep} + 1$)
        ADD($ListSensor_{ON}$, $Sensor_{Hexagon}$)
        TURN ON $Sensor_{Hexagon}$
      }
      IF( $Intruder_{NewLocation} \in$ SensorListON)
```

```
        {
          $\text{Intruder}_{\text{CurrentLocation}}$ = $\text{Intruder}_{\text{NewLocation}}$
        {
        ELSE
        {
            $\text{Intruder}_{\text{NewLocation}}$ = FindIntruder($\text{Intruder}_{\text{CurrentLocation}}$)
            $\text{Intruder}_{\text{CurrentLocation}}$ = $\text{New}_{\text{IntruderLocation}}$
          }
      }
    }
  }
```

8. THE TACTIC WEIGHT BASED ALGORITHM

8.1. Overview

The Tactic Weight Based Algorithm uses the movement probabilities generated from the terrain complexity (weight) discussed in chapter 3 to schedule sensors and track intruders in the wireless sensor network. The movement probability is the probability of movement of the intruder from a particular hexagon to all the adjacent hexagons. The **TWBA** sorts the adjacent sensor hexagons in ascending order by the movement probability. The user specifies the percentage of adjacent sensors that have to be turned ***ON***. The **TWBA** turns ***ON*** the percentage of adjacent sensors specified by the user with the highest probability sensors being turned on first. The objective of the **TWBA** algorithm is to provide effective intruder tracking while increasing the sensor network's life time.

As input, **TWBA** needs information about the present location of the intruder ($\boldsymbol{Intruder_{CurrentLocotion}}$), the movement probabilities ($\boldsymbol{Movement_{probability}}$) and percentage of adjacent sensors to turn ON ($Percentage_{ON}$). **TWBA** maintains a list ($\boldsymbol{ListSensors_{ON}}$) of probable locations ($\boldsymbol{Sensor_{Hexagon}}$) the intruder would move in the next step. The function $\boldsymbol{Adjacent(Intruder_{CurrentLocation}}$ returns a set of all $\boldsymbol{Sensor_{Hexagon}}$ adjacent to the current intruder location. Initially $\boldsymbol{ListSensors_{ON}}$ is empty. Table 8.1. lists all the abbreviations used in the **TWB** algorithm

8.2. Algorithm

Step 1: Figure 8.1. shows the pseudo code for step 1. The intruder is first detected on the upper boundary ($\boldsymbol{Upper_{Boundary}}$). The **TWBA** is initialized on the first detection. As an input, the algorithm needs current intruder location

(***$Intruder_{CurrentLocation}$***), the movement probabilities (***$Movement_{probability}$***) and the percentage of sensors to be turned ON (***$Percentage_{ON}$***). **TWBA** then checks to see if ***$ListSensors_{ON}$*** is empty. If ***$ListSensors_{ON}$*** is not empty, all the *$Sensor_{Hexagon}$* listed in ***$ListSensors_{ON}$*** are turned OFF and the ***$ListSensors_{ON}$*** is cleared.

```
BEGIN
{
  $ListSensors_{ON} = \Phi$
  WHILE($Intruder_{CurrentLocation} \notin Boundary_{Lower}$)
  {
     IF($ListSensors_{ON} \neq \Phi$)
     {
        FOREACH $Sensor_{Hexagon} \in ListSensors_{ON}$
        TURN OFF $Sensor_{Hexagon}$
     }
    $ListSensors_{ON} = \Phi$
```

Figure 8.1. The Tactic Weight Based Algorithm: STEP 1

Step 2: Figure 8.2. shows the pseudo code for step 2. Create a list of sensors adjacent to the current intruder location and sort the list in ascending order by

```
ADJACENT = SORT(ADJACENT($Intruder_{CurrentLocation}$),
                              $Movement_{probability}$, Ascending)
NumberOfSensorsON=($Percentage_{ON}$ + ADJACENT. Count) / 100
Count = 0
While(NumberOfSensorsON > 0)
 {
    $Sensor_{Hexagon}$ = ADJACENT.getElementat(Count)
    ADD($ListSensor_{ON}$, $Sensor_{Hexagon}$)
    TURN ON $Sensor_{Hexagon}$
    NumberOfSensorsON = NumberOfSensorsON – 1
    Count = Count + 1
 }
```

Figure 8.2. The Tactic Weight Based Algorithm: STEP 2

movement probability. Convert the percentage of adjacent sensors to turn on into the number of adjacent sensors to turn on. Turn on the adjacent sensors.

Step 3: Figure 8.3. shows the pseudo code for step 3. In the next step, the intruder will move from his current location to a new location. If the new intruder location is a sensor hexagon and the sensor hexagon was turned on in STEP 2, the

```
            $Intruder_{CurrentLocation}$ = $Intruder_{NewLocation}$
        }
        ELSE
        {
            $Intruder_{NewLocation}$ = FindIntruder($Intruder_{CurrentLocation}$)
            $Intruder_{CurrentLocation}$ = $New_{IntruderLocation}$
        }
    }
  }
}
```

Figure 8.3. The Tactic Weight Based Algorithm: STEP 3

intruder has been successfully tracked. If the intruder has been successfully tracked, STEP 1 through STEP 3 is repeated with the new intruder location. If the intruder has not been successfully tracked, that is, the intruder has moved to a sensor hexagon which was not turned on in the STEP 2, the recovery algorithm to trace the location of intruder is called. When the new location of intruder is traced, STEP 1 through STEP 3 is repeated with new intruder location. The algorithm is terminated if the intruder moves to a location on the lower boundary.

Table 8.1. Abbreviations Used in the Tactic Weight Based Algorithm

Abbreviation	**Meaning**
$\text{Intruder}_{\text{CurrentLocation}}$	Current location of the intruder
$\text{Intruder}_{\text{NewLocation}}$	New intruder location
$\text{Movement}_{\text{probability}}$	Set containing movement probabilities from a sensor hexagon to all its adjacent sensor hexagons
$\text{ListSensors}_{\text{ON}}$	Set of probable locations ($\text{Sensor}_{\text{Hexagon}}$) where the intruder can move in the next step
SORT(ADJACENT($\text{Intruder}_{\text{CurrentLocation}}$), $\text{Movement}_{\text{probability}}$, Ascending)	Sort the set of adjacent sensors in the ascending order by $\text{Movement}_{\text{Probability}}$.
Adjacent (S_i)	{ $\text{Sensor}_{\text{Hexagon}}$ | $\forall$ $\text{Sensor}_{\text{Hexagon}}$ are adjacent to $\text{Sensor}_{\text{Hexagon}}$ S_i}

8.3. Pseudo Code: The Tactic Weight Based Algorithm

```
INPUT: Current location of intruder (IntruderCurrentLocation)
        Percentage of adjacent sensors to turn ON (PercentageON)
        Movement probabilities (Movementprobability)
1 BEGIN
```

```
{
ListSensorsON = $\Phi$
WHILE(IntruderCurrentLocation $\notin$ BoundaryLower)
{
   IF(ListSensorsON $\neq \Phi$)
   {
      FOREACH SensorHexagon $\in$ ListSensorsON
      TURN OFF SensorHexagon
   }
   ListSensorsON = $\Phi$
   //Sort the sensors adjacent to the current location in
   //ascending order of movement probability
   ADJACENT = SORT(ADJACENT(IntruderCurrentLocation),
                                            Movementprobability, Ascending)
   NumberOfSensorsON=(PercentageON + ADJACENT. Count) / 100
   Count = 0
   While(NumberOfSensorsON > 0)
    {
        SensorHexagon = ADJACENT.getElementat(Count)
        ADD(ListSensorON, SensorHexagon)
        TURN ON SensorHexagon
        NumberOfSensorsON = NumberOfSensorsON – 1
        Count = Count + 1
```

```
            }
            IF( IntruderNewLocation ∈ SensorListON)
            {
                IntruderCurrentLocation = IntruderNewLocation
            }
            ELSE
            {
                IntruderNewLocation = FindIntruder(IntruderCurrentLocation)
                IntruderCurrentLocation = NewIntruderLocation
            }
        }
    }
}
```

9. COMMUNICATION ALGORITHM

9.1. Overview

We have created a simple communication algorithm to use with the sensor scheduling and tracking algorithms discussed in the previous sections. It is to be noted that the major area of study in this research is the sensor scheduling and tracking algorithms and not the communication algorithm.

The sensor scheduling and tracking algorithm is run on the base station, but the communication algorithm runs on individual sensors. The sensors in the sensor network maintain a list of all the adjacent sensors and the amount of energy remaining at each adjacent sensor. A sensor knows the position of an adjacent sensor relative to the base station and itself by looking at the adjacent sensor index. ***Near neighbors*** are the adjacent sensor hexagons that share physical boundary with the current sensor hexagon.

9.2. Algorithm

Step 1: Examine all the near neighbors and find the near neighbors whose index is greater than the index of the current sensor and which has the most energy remaining.

Step 2: If a near neighbor is found in Step 1, communicate to the near neighbor, otherwise, examine at all the adjacent sensors which are not near neighbors but their index is greater than the index of the current sensor hexagon. Find the sensor hexagon with the most energy remaining.

Step 3: Communicate to the adjacent sensor found in Step 2. Continue Step 1 through Step 3 until the information has been communicated to the base station.

The communication happens in multiple hops. At each hop the sensor communicating attaches its current energy level to the information packet. The base station maintains and constantly updates the energy information of each sensor. The base station broadcasts the energy information and the list of adjacent sensors to every sensor in the sensor network at the end of each intruder tracking.

10. THE TACTIC EVALUATION AND SIMULATION TOOL

10.1 Overview

We have created a Tactic Evaluation and Simulation Tool (TEST) to effectively analyze and simulate the performance of the tactics. The TEST enables visualization of all the sensor scheduling and intruder tracking algorithms and the communication algorithm discussed in the previously. The user interface of test is shown in Figure 10.1. The simulator consists of two panes; the left pane is used to draw the

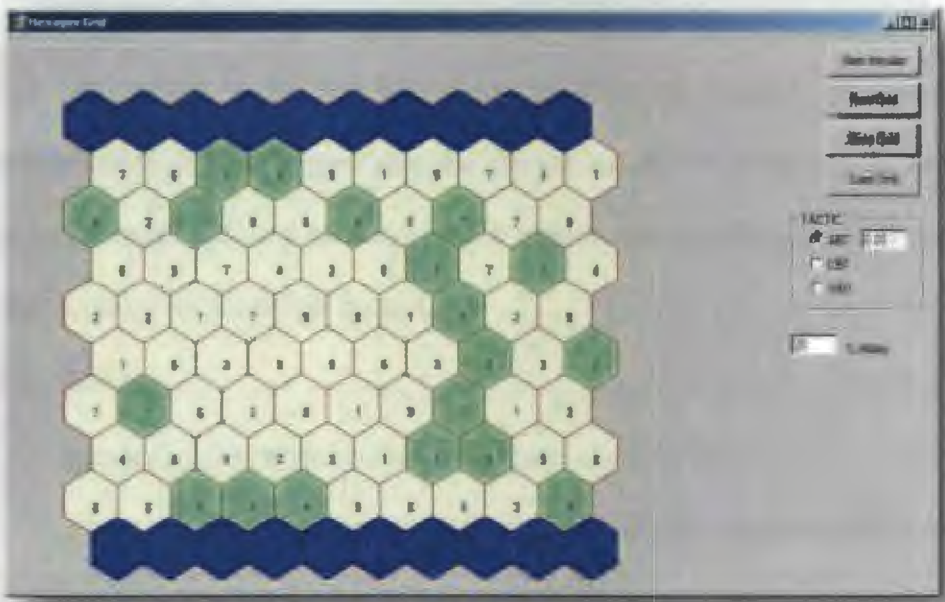


Figure 10.1. The Tactic Evaluation and Simulation Tool

simulation and the right pane consists of controls used to regulate the simulation. The tool has been created using C Sharp.

By default the 10 × 10 sensor network grid is created on the left pane. On the right pane, a user can introduce a new intruder into the sensor network by pressing the "New Intruder" button. The "New Grid" button is used to generate a fresh sensor network on the left pane. The "Save Grid" button can be used to save the currently drawn sensor network. The "Load Grid" button can be used to load an already saved sensor network. In the tactic group, one can choose the tactic you will be using to schedule the sensor and track the intruders. When one chooses the "**TABA**" option in the sensor group the Tactic Association Based Algorithm will be activated. One can specify the minimum threshold value needed for the **TABA** in the text box which becomes visible when one chooses the "**TABA**" option. When one chooses the "**TCBA**" option in the sensor group, the Tactic Case Based Algorithm will be activated. When one chooses the "**TWBA**" option in the sensor group, the Tactic Weight Based Algorithm will be activated. One can specify the percentage of sensors to turn ON when using **TWBA** in the text box which becomes visible when you choose the "**TWBA**" option. In the "% Holes" text box one can specify the percentage of holes that should be contained in a sensor network before one can generate a new sensor network using the "New Grid" button.

10.2 Classes

Following is a brief description of function of each class:

- ActionIntrudeMissed: This class contains the algorithm which is called when the intruder is not detected by a tactic.
- Adjacent: It consists of method that generates a list of sensor hexagons adjacent to a given hexagon.
- BitVector: This call is used to generate support of a hexagon in the sensor network.
- ClearGrid: This class is used to clean the drawing on the right plane after each draw.

- Dijkstra: This class contains methods to calculate the shortest path for an intruder through the sensor network.
- GeneratePath: This class contains methods used to generate an intruder path through the given sensor network.
- Hexagon: This class contains the details of the basic elements of hexagon drawn on the left pane.
- HexagonViewer: This class contains most of the code that draws the form and other elements on the form.
- HexGrid: This class contains all the setup algorithms needed for the tactics to work. It also contains the code that controls the various elements of the sensor network.
- Neighbours: This class consists of a single method that determines if a hexagon shares a boundary with another hexagon.
- OperationCost: This class contains a method that calculates the energy spent by a sensor in tracking an intruder and communicating to an adjacent sensor.
- Probability: This class contains a method that generates the movement probabilities for an intruder using the "Terrain Complexity" of the hexagon.
- RandomPathBiased: This class generates historical paths that will be used as the base data by tactics.
- CaseBasedTactic: This class contains the Tactic Case Based Algorithm

- AssociationBasedTactic: This class is the Tactic Association Based Algorithm.
- WeightBasedTactic: This class is the Tactic Weight Based Algorithm.
- WeightedGraph: This class is the supporting class for the Dijkstra class.

11. EXPERIMENTATION

11.1. Overview

In this section we discuss the experimental design, experiments and evaluate the performance of the **TCBA**, **TABA** and **TWBA** algorithms. Before we proceed with the discussion of the experimental design and the experiments, it is important to understand the metrics we will be using to evaluate the performance and the parameters that affect the performance of **TCBA**, **TABA** and **TWBA**.

Performance Metrics:

For evaluating the performance of the scheduling and tracking algorithms we consider the following performance metrics:

1. Accuracy of prediction
2. Energy efficiency (Sensor network life time)
3. Scalability

The performance of **TCBA** depends on the following independent parameters:

1. Number of historical paths
2. Percentage of holes in the sensor network
3. Communication cost
4. Number of simultaneous intrusions
5. Size of the sensor grid

The performance of **TABA** depends on the following independent parameters:

1. Number of historical paths
2. Percentage of holes in the sensor network
3. The minimum threshold value

4. Communication cost
5. Number of simultaneous intrusions
6. Size of the sensor grid

The performance of **TWBA** depends on the following independent parameters:

1. Percentage of holes in the sensor network
2. Communication cost
3. Number of simultaneous intrusions
4. Size of the sensor grid.
5. Percentage of adjacent sensors to turn ON

11.2. Experimental Design

We use sensor scheduling and tracking simulator, discussed in the previous section, as a tool for experimentation and analysis. A sensor grid size of 10*10 is used for all the experiments unless otherwise specified. Terrain complexity is randomly assigned to the hexagons when the grid is generated. The proportion of holes and sensors in the sensor grid is subject to individual experiments. The placement of sensors in the sensor grid is non-deterministic. The battery life of a sensor is 100 units. The cost of surveillance on the sensor for a single iteration is 1 unit. The cost of communicating to the adjacent sensor is 10 units. Once the sensor has depleted 90 units of battery power, it is rendered useless. In this case, we say that the sensor is dead and hexagon containing the dead sensor is treated as a hole.

The hexagon on the upper and the lower boundary contain sensors. These hexagons have unlimited power supply and are always in an ON state. The intrusion starts at the upper boundary and ends at the lower boundary. The communication process used has been discussed in chapter 9. The sensor scheduling and tracking algorithms used are discussed in chapter **Error! Reference source not found.**, 7 and REF _Ref264495838 \w \h * MERGEFORMAT **Error! Reference source not found.**.

11.3. Effect of Varying Minimum Threshold Value on the Detection of Intruder When Using TABA

In TABA the minimum threshold value is compared with the confidence of intruder moving from the present sensor location to each adjacent sensor location. Only those adjacent sensors are turned on where the confidence of moving from the present sensor location to the adjacent sensor location is greater than the minimum threshold value. Chart 11.1. displays the effect of varying minimum threshold value on the detection of intruder when using TABA

- *Independent Variable*: Minimum threshold value
- *Independent variable values*: 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%.
- *Number of replications for each threshold value* : 50
- *Dependent Variables*:
  - Grid Size: 10*10
  - Percentage of holes: 0%
  - Sensors Die: No
  - Line of sight: 3
  - Path for path/Boundary cell: 500

Observations: For minimum threshold value of 0%, we get 100% detection. With an increase in the minimum threshold value, the average number of detections

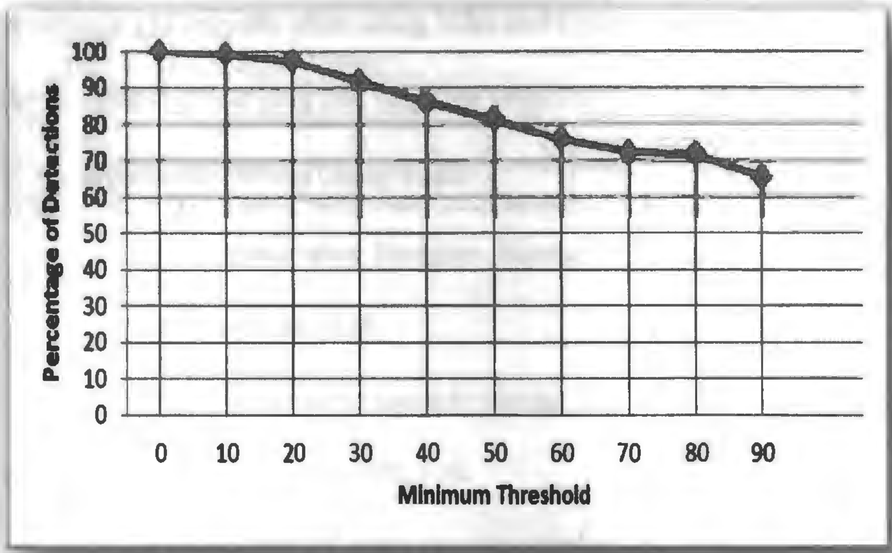


Chart 11.1. Effect of Varying Minimum Threshold Value on the Detection of Intruder When Using TABA

consistently decreases. For minimum threshold value of 100% we still get at least 50% detection.

Interpretation: If the minimum threshold is 0%, it means that all the adjacent sensors to which the probability of movement of the intruder greater than 0% is turned ON. Thus, we have 100% detection. As the minimum threshold value is increased, the number of sensors turned ON decreases. Hence, the average detection rate also decreases. The point worth noting is that with the minimum threshold of 100%, we get the average detection of about 50%.

11.4. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TCBA and TABA

Both **TCBA** and **TABA** use historical intrusion path data for predicting intruder location. Neither of these algorithms can be used if no historical path data is available. A rich set of historical paths can effectively increase the detection rate of an intruder.

Chart 11.2. and Chart 11.3. show the effect of varying the number of historical paths on the detection of intruder when using TCBA and TABA respectively

11.4.1. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TCBA

- *Independent Variable*: Number of historical paths starting per boundary cell
- *Independent variable values*: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100, 200, 500.
- *Number of replications per Independent variable values*: 50
- *Dependent Variables*:
  - Grid Size: 10*10
  - Percentage of holes: 0%
  - Sensors Die: No
  - Line of sight: 3

Observation: With 1 historical path per boundary cell we get at least 50% detection. As paths per boundary cell increases, the detection rate also increases. With 30 or more historical path per boundary cells, we get at least 90% detection. With 500 historical paths per boundary cells, we get almost 100% detection rate.

Interpretation: **TCBA** compares the intruder movement pattern with the historical paths stored in the database. If fewer historical paths are available, **TCBA** has fewer cases to compare with and thus the probability of intruder being missed is high.

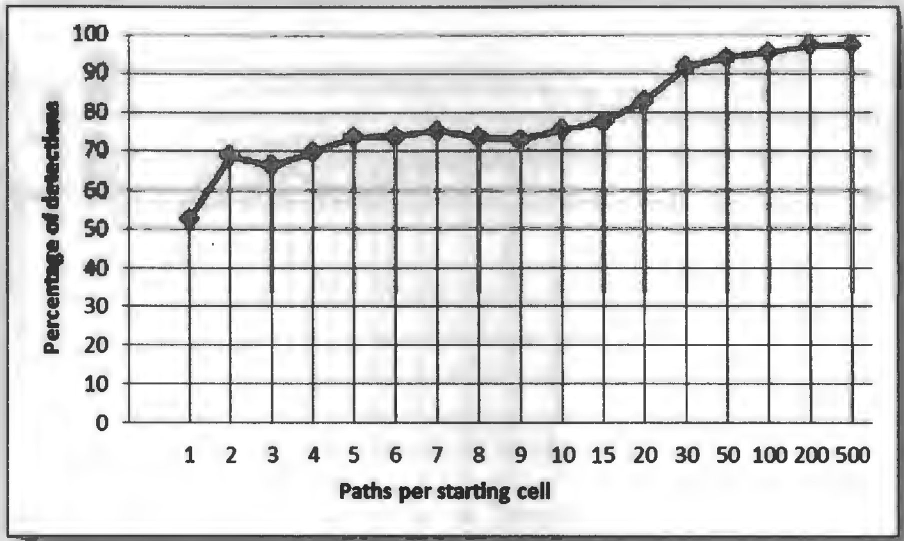


Chart 11.2. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TCBA

11.4.2. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TABA

- *Independent Variable: Number of historical paths starting per boundary cell*
- *Independent variable values: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100, 200, 500.*
- *Number of replications per Independent variable values: 50*
- *Dependent Variable Values:*
    - Grid Size: 10*10
    - Percentage of holes: 0%
    - Sensors Die: No
    - Line of sight: 3
    - Path for path/Boundary cell: 500

- Minimum Threshold: 0%

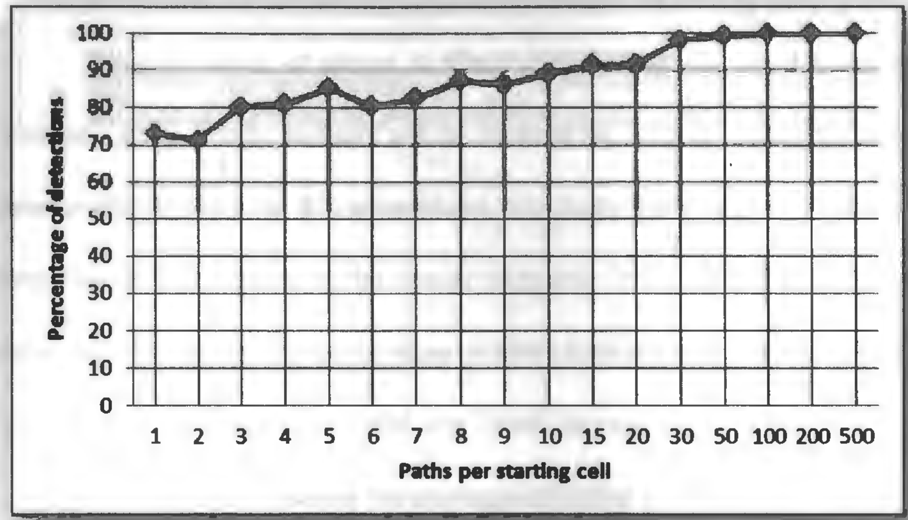


Chart 11.3. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TABA

Observation: With one historical path per boundary cell we get at least 70% detection. As paths per boundary cell increases the detection rate also increases. With 20 or more historical path per boundary cells we get at least 90% detection and with 100 historical paths per boundary cells we get almost 100% detection rate.

Interpretation: With fewer historical paths, the confidences generated may not represent the actual movement patterns of the intruder. Hence, the chance of missing the intruder is high. As the number of historical path increases, the confidences generated represent the actual trend of the intruder's movement pattern and consequently the chance of correctly detecting the intruder increases.

11.5. Effect of Varying Percentage of Holes in the Sensor Network on the Detection of Intruder When Using TCBA, TABA and TWBA

The distribution of sensor in the sensor network is non-deterministic. Not all hexagons in the sensor network will be covered by sensors leading to the presence of holes in the network. In this experiment, we study the effect of holes on the rate of detection of the intruder in the sensor network. Chart 11.4, Chart 11.5, Charts 11.6 show the effect of varying percentage of holes in the sensor network on the detection of intruder when using TCBA, TABA and TWBA respectively

11.5.1. Effect of Varying Percentage of Holes in the Sensor Network on the Detection of Intruder When Using TCBA

- *Effect of varying percentage of holes in the sensor network on the*
- *dent Variable: Percentage of holes in the sensor network*
- *Independent variable values: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100*
- *Number of replications per Independent variable values: 500*
- *Dependent Variable Values:*
    - Grid Size: 10*10
    - Sensors Die: No
    - Line of sight: 3
    - Path for path/Boundary cell: 500

Observation: With 0% holes in the sensor network, we get 100% intruder detection. As the number of holes in the sensor network increase, the detection rate

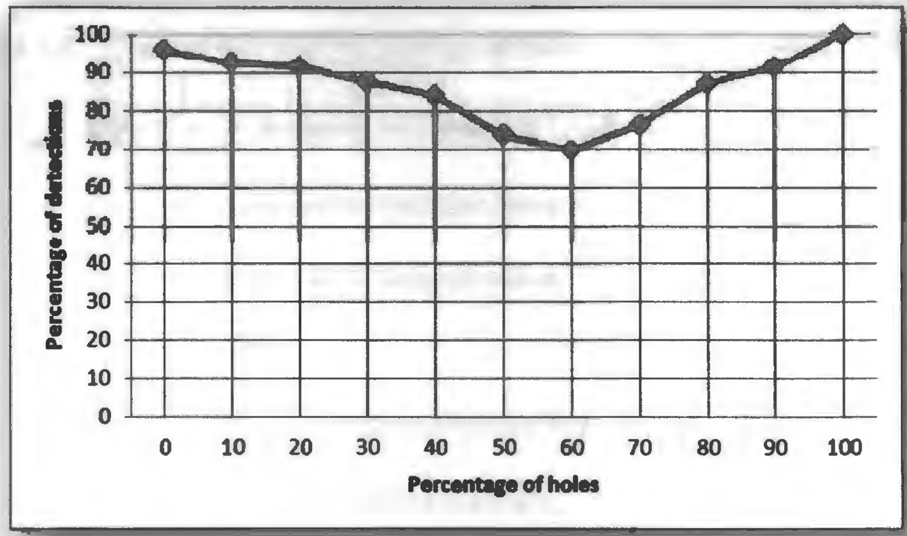


Chart 11.4. Effect of Varying Percentage of Holes in the Sensor Network on the Detection of Intruder When Using TCBA

decreases. We observe minimum detection rate of 70% with 60% holes in the sensor network. The detection gradually increases from 60% holes to 100% holes in the sensor network.

Interpretation: With 0% holes, the intruder in the sensor network has a maximum of 6 adjacent hexagons to choose from, when deciding where to move. As the percentage of holes in the sensor network increase, the number of adjacent hexagons increases. Consequently there is a greater number of adjacent hexagons to choose from when deciding the next move. Thus, the probability that the intruder will be missed increases. With percentage of holes greater than 60%, there is a high probability the intruder will only be detected at the upper boundary and the lower boundary. In this case most of the intruder path will be though holes. With 100% holes in the sensor network, it is expected that the intruder will only be detected at the upper boundary and the lower boundary and since the sensor nodes at the boundary are always ON we get 100% detection.

11.5.2. Effect of Varying Percentage of Holes in the Sensor Network on the Detection of Intruder When Using TABA

- *Independent Variable: Percentage of holes in the sensor network*
- *Independent variable values: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100*
- *Number of replications per Independent variable values: 50*
- *Dependent Variable Values:*
    - Grid Size: 10*10
    - Sensors Die: No
    - Line of sight: 3
    - Path for path/Boundary cell: 500
    - Minimum Threshold: 0%

Observation: We get 100% detection with any percentage of holes in the sensor network.

Interpretation: The minimum threshold value is 0%. Hence, almost all the sensor nodes adjacent to the current position of the intruder are turned ON.

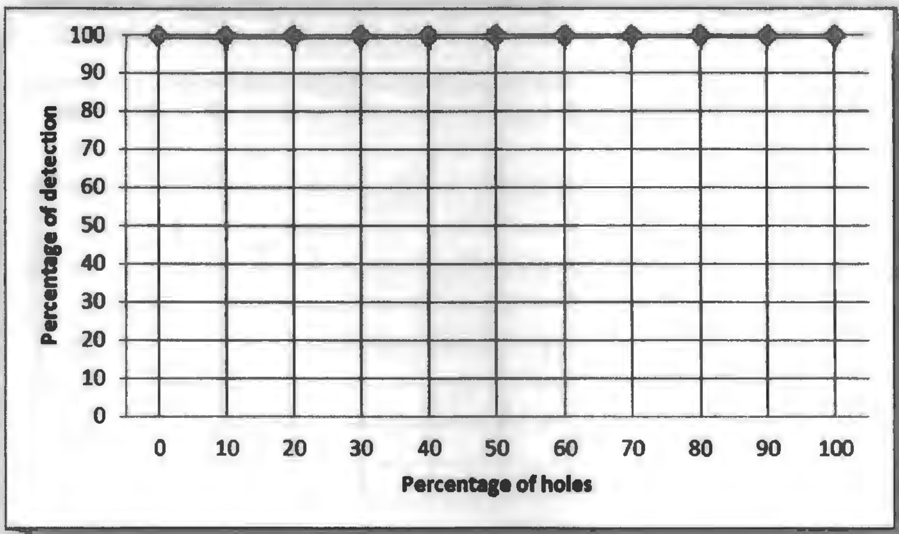


Chart 11.5. Effect of Varying Percentage of Holes in the Sensor Network on the Detection of Intruder When Using TABA

11.5.3. Effect of Varying Percentage of Holes in the Sensor Network on the Detection of Intruder When Using TWBA

- *Independent Variable: Percentage of holes in the sensor network*
- *Independent variable values: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100*
- *Number of replications per Independent variable values: 50*
- *Dependent Variable Values:*
    - Grid Size: 10*10
    - Sensors Die: No
    - Line of sight: 3
    - Path for path/Boundary cell: 500

- Maximum load: 100%

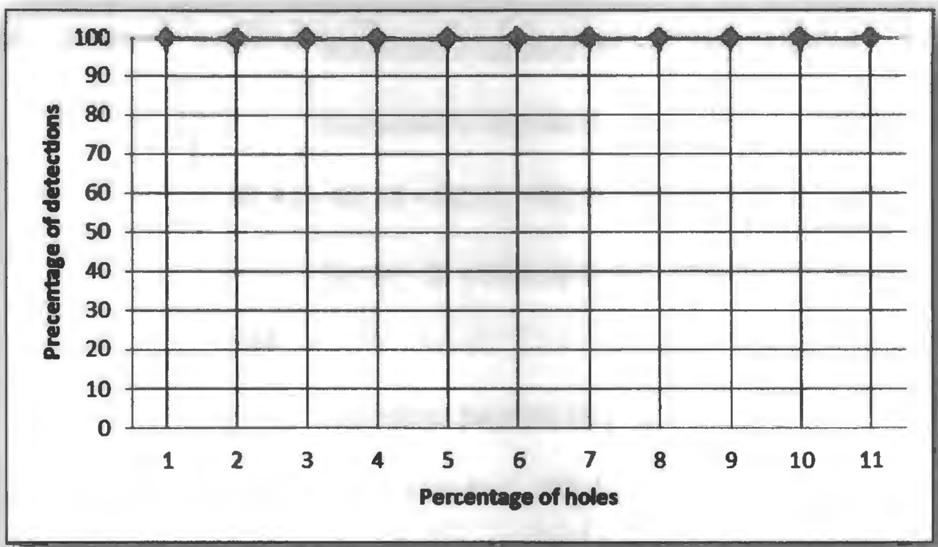


Chart 11.6. Effect of Varying Percentage of Holes in the Sensor Network on the Detection of Intruder When Using TWBA

Observation: We get 100% detection with any percentage of holes in the sensor network.

Interpretation: The maximum load is 100%. Hence all the sensor nodes adjacent to the current position of the intruder are turned ON.

11.6. Effect of Sensors Dying on the Detection of Intruder When Using TABA, TCBA and TWBA

The battery life of a sensor is 100 units. The cost of surveillance on the sensor for a single iteration is 1 unit. The cost of communicating to the adjacent sensor is 10 units. Once the sensor has depleted 90 units of battery power, it is rendered useless. In this case we say that the sensor is dead and hexagon containing the dead sensor is treated as a hole. Chart 11.7., Chart 11.8., Chart 11.9. show the effect of sensors dying on the detection of intruder when using TABA, TCBA and TWBA respectively.

11.6.1. Effect of Varying the Number of Sensors Dying on the Detection of Intruder When Using TCBA

- *Independent Variable*: Percentage of sensors dead
- *Independent variable range*: 0 - 10, 11 - 20, 21 - 30, 31 – 40, 41 – 50, 51 – 60, 61 – 70, 71 – 80, 81 – 90, 91 - 100
- *Number of replications per independent variable values*: 500
- *Dependent Variable Values*:
    - Grid Size: 10*10
    - Sensors Die: Yes
    - Line of sight: 3
    - Path for path/Boundary cell: 500

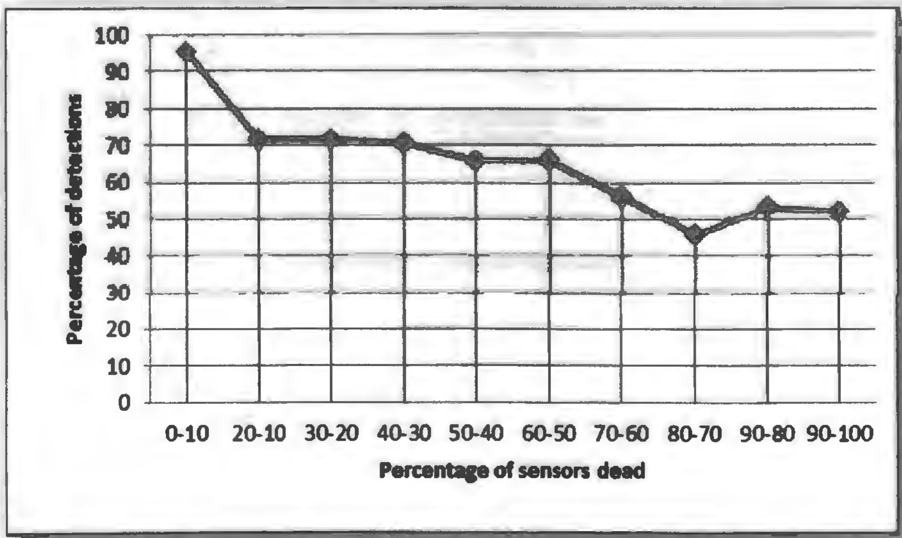


Chart 11.7. Effect of Varying the Number of Sensors Dying on the Detection of Intruder When Using TCBA

Observations: For 0-10% holes in the sensor network we get nearly 95% detection. As the percentage of holes increases, the percentage of detections gradually decreases. For 90-100% holes, we get at least 50% detection.

Interpretation: With 0% holes, the intruder in the sensor network has a maximum of 6 adjacent hexagons to choose from when deciding where to move. As the percentage of holes in the sensor network increase, the number of adjacent hexagons increases. Consequently there are a greater number of adjacent hexagons to choose from when deciding the next move. Thus, the probability that the intruder will be missed increases.

11.6.2. Effect of Varying the Number of Sensors Dying on the Detection of Intruder When Using TABA

- *Independent Variable*: Percentage of sensors dead
- *Independent variable range*: 0 - 10, 11 - 20, 21 - 30, 31 – 40, 41 – 50, 51 – 60, 61 – 70, 71 – 80, 81 – 90, 91 - 100
- *Number of replications per Independent variable values*: 500
- *Dependent Variable Values*:
  - Grid Size: 10*10
  - Sensors Die: Yes
  - Line of sight: 3
  - Path for path/Boundary cell: 500
  - Minimum Threshold: 0%

Observations: For 0-10% of holes in the sensor network we get nearly 100% detection. As the percentage of holes increases, the percentage of detections gradually decreases. For 90-100% we get at least 50% detection.

Interpretation: With 0% holes, the intruder in the sensor network has a maximum of 6 adjacent hexagons to choose from, when deciding where to move. As the percentage of holes in the sensor network increase, the number of adjacent hexagons increases. Consequently there are a greater number of adjacent hexagons to choose from when deciding the next move. Thus, the probability that the intruder will be missed increases.

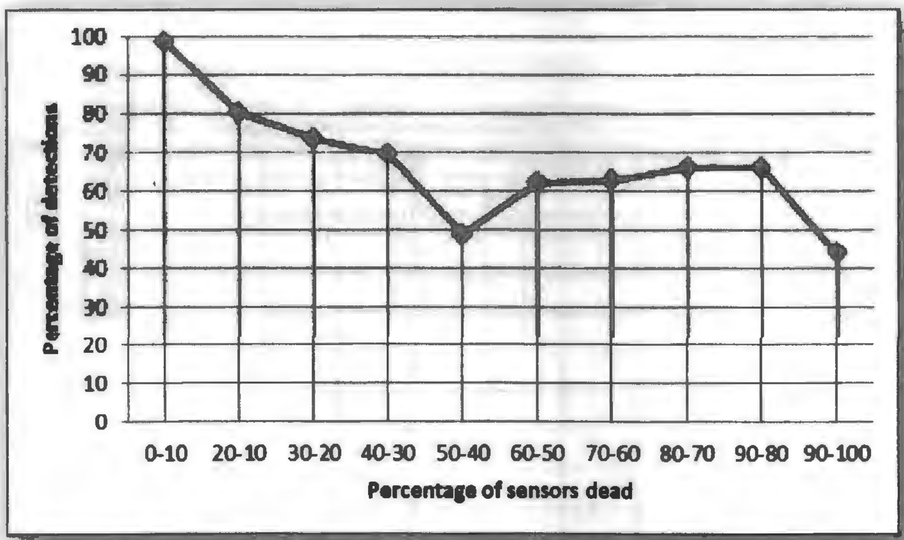


Chart 11.8. Effect of Varying the Number of Sensors Dying on the Detection of Intruder When Using TABA

11.6.3. Effect of Varying the Number of Sensors Dying on the Detection of Intruder When Using TWBA

- *Independent Variable*: Percentage of sensors dead

- *Independent variable range*: 0 - 10, 11 - 20, 21 - 30, 31 – 40, 41 – 50, 51 – 60, 61 – 70, 71 – 80, 81 – 90, 91 - 100
- *Number of replications per Independent variable values*: 500
- *Dependent Variable Values*:
    - Grid Size: 10*10
    - Sensors Die: Yes
    - Line of sight: 3
    - Path for path/Boundary cell: 500
    - Maximum load: 100%

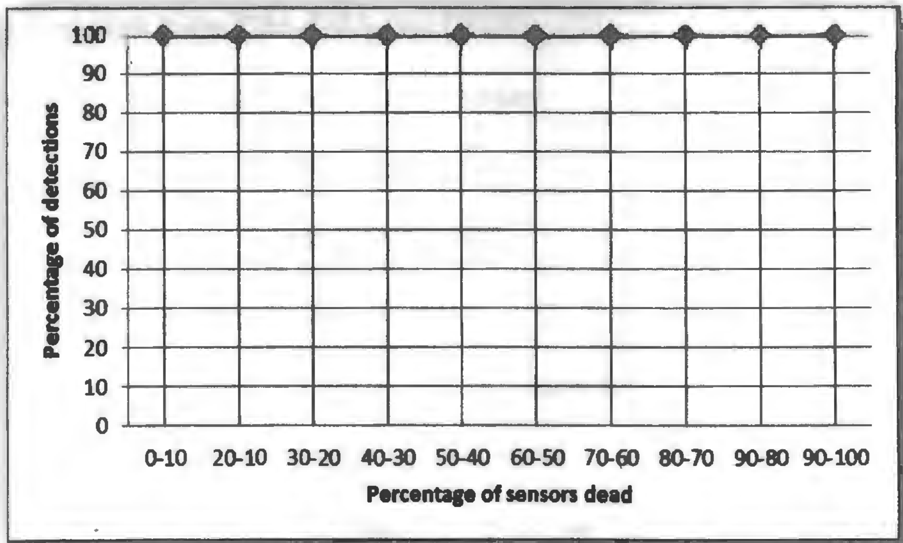


Chart 11.9. Effect of Varying the Number of Sensors Dying on the Detection of Intruder When Using TWBA

Observation: We get 100% detection with any percentage of sensors dead in the sensor network.

Interpretation: The maximum load is 100%. Hence all the sensor nodes adjacent to the current position of the intruder are turned **ON**.

11.7. Life Time of Sensor Network With no Communication Cost

In this experiment, we compare the life time of the sensor network when using different sensor scheduling and tracking algorithms. We say a sensor network is operational if less than 60% of hexagons in the network are holes. The battery life of a sensor is 100 units. The cost of surveillance on the sensor for a single iteration is 1 battery unit. Once the sensor has depleted 90 units of battery power it is rendered useless. In this case we say that the sensor is dead and hexagon containing the dead sensor is treated as a hole. Life time of the sensor network is measured by number of scheduling done in the sensor network. Chart 11.10 shows the life time of a 10*10 sensor grid when using TCBA, TABA, and TWBA

- *Dependent Variable Values*:
    - Grid Size: 10*10
    - Sensors Die: Yes
    - Line of sight: 3
    - Path for path/Boundary cell: 500
    - Maximum load: 100%
    - Minimum threshold: 0%
    - Percentage of sensors dead: 100
    - Maximum load: 100%
    - Communication cost: No

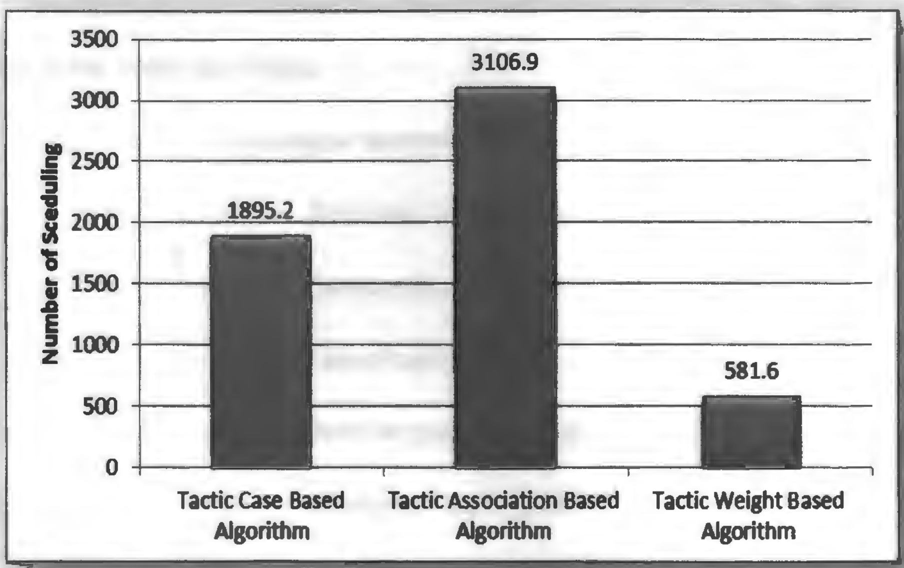


Chart 11.10. Life Time of Sensor Network With no Communication Cost

Observation: When using **TCBA**, the sensor network is scheduled for 1895 times, on an average. While using **TABA**, the sensor network is scheduled for 3106 times, on an average. While using **TWBA**, the sensor network is scheduled for 581 times, on an average.

11.8. Life Time of a Sensor Network With Communication Cost

In this experiment, we compare the life time of the sensor network when using different sensor scheduling and tracking algorithms. We assume a sensor network is operational if less than 60% of hexagons in the network are holes. The battery life of a sensor is 100 units. The cost of surveillance on the sensor for a single iteration is 1 battery unit. The cost of communication is 10 battery units. Once the sensor has depleted 90 units of battery power, it is rendered useless. In this case, we assume that the sensor is dead and hexagon containing the dead sensor is treated as a hole. Life time of the sensor network is measured by number of scheduling done in the sensor network. Chart 11.11. shows the life time of a sensor network with communication cost in TCBA, TABA and TWBA.

- *Dependent Variable Values*:
  - Grid Size: 10*10
  - Sensors Die: Yes
  - Line of sight: 3
  - Path for path/Boundary cell: 500
  - Maximum load: 100%
  - Minimum threshold: 0%
  - Percentage of sensors dead: 100
  - Maximum load: 100%
  - Communication cost: Yes

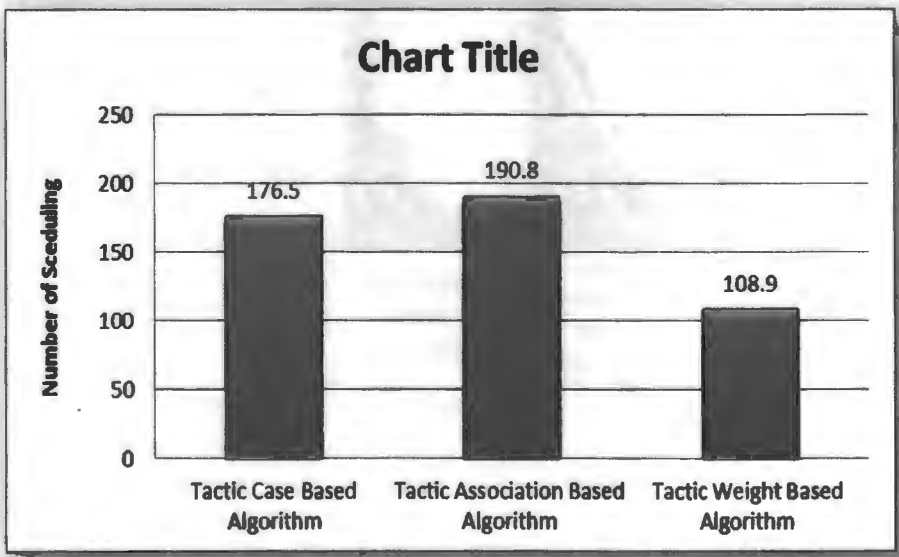
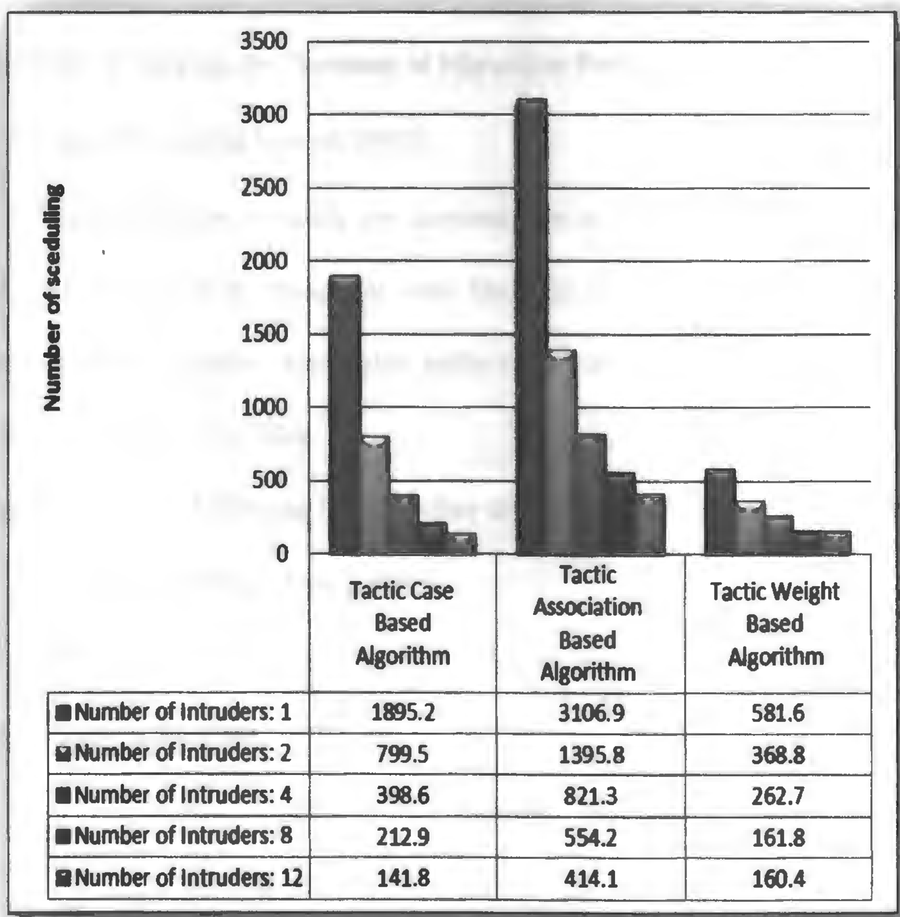


Chart 11.11. Life Time of Sensor Network With Communication Cost

Observation: When using **TCBA**, the sensor network is scheduled for 176 times, on an average. When using **TABA**, the sensor network is scheduled for 190 times, on an average. When using **TWBA**, the sensor network is scheduled for 108 times, on an average.

11.9. Lifetime of the Sensor Network When Number of Intruders is Varied

In this experiment, we compare the life time of the sensor network when using different sensor scheduling and tracking algorithms. Chart 11.12. shows the life time of a sensor network when the number of intrudes simultaneously present in the grid is



	Tactic Case Based Algorithm	Tactic Association Based Algorithm	Tactic Weight Based Algorithm
■ Number of Intruders: 1	1895.2	3106.9	581.6
■ Number of Intruders: 2	799.5	1395.8	368.8
■ Number of Intruders: 4	398.6	821.3	262.7
■ Number of Intruders: 8	212.9	554.2	161.8
■ Number of Intruders: 12	141.8	414.1	160.4

Chart 11.12. Life Time of the Sensor Network for Different Number of Intruders

changed. We assume a sensor network is operational if less than 60% of hexagons in the network are holes. The battery life of a sensor is 100 units. The cost of surveillance on the sensor for a single iteration is 1 battery unit. Once the sensor has depleted 90 units of battery power, it is rendered useless. In this case, we say that the sensor is dead and hexagon containing the dead sensor is treated as a hole. Life time of the sensor network is measured by number of scheduling done in the sensor network.

Observation: For all the three algorithms, as the number of simultaneous intruders increases the lifetime of the sensor network decreases.

11.10. Effect of Varying the Number of Historical Paths on the Detection of Intruder for a Grid Size of 50*50

In this experiment we study the performance of **TCBA** and **TABA** for a sensor network of grid size 50*50 hexagonal cells. Chart 11.13. and Chart 11.14 show the effect of varying the number of historical paths on the detection of intruder for a grid size of 50*50 for TCBA and TABA.

11.10.1. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TCBA

Chart 11.3 shows the effect of varying the number of historical paths on the detection of intruder when using TCBA.

- *Independent Variable*: Number of historical paths starting per boundary cell
- *Independent variable values*: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100, 200, 500.

- *Number of replications per Independent variable values*: 50
- *Dependent Variables*:
  - Grid Size: 50*50
  - Percentage of holes: 0%
  - Sensors Die: No
  - Line of sight: 3

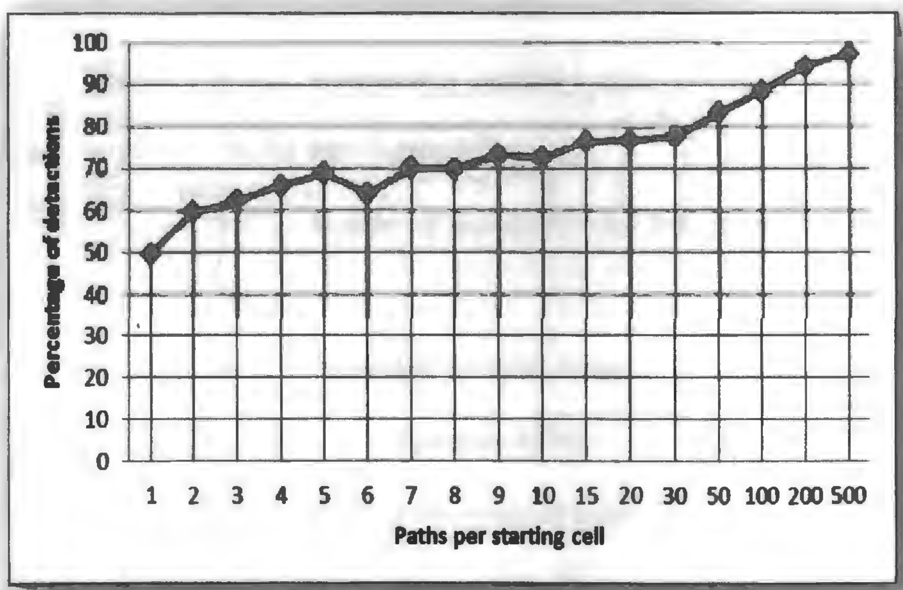


Chart 11.13. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TCBA

Observation: With 1 historical path per boundary cell we get at least 50% detection. As paths per boundary cell increases, the detection rate increases. With 30 or more historical path per boundary cells, we get at least 90% detection. With 500 historical paths per boundary cells, we get nearly 100% detection rate.

Interpretation: **TABA** compares the intruder movement pattern with the historical paths stored in the database. If fewer historical paths are available, **TABA**

has fewer cases to compare with and thus the probability of intruder being missed is high.

11.10.2. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TABA

Chart 11.14 shows the effect of varying the number of historical paths on the detection of intruder when using TABA.

- *Independent Variable*: Number of historical paths starting per boundary cell
- *Independent variable values*: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 50, 100, 200, 500.
- *Number of replications per Independent variable values*: 50
- *Dependent Variable Values*:
    - Grid Size: 50*50
    - Percentage of holes: 0%
    - Sensors Die: No
    - Line of sight: 3
    - Path for path/Boundary cell: 500
    - Minimum Threshold: 0%

Observation: With one historical path per boundary cell we get at least 70% detection. As paths per boundary cell increases, the detection rate also increases. With 20 or more historical path per boundary cells, we get at least 90% detection and with 100 historical paths per boundary cells we get nearly 100% detection rate.

Interpretation: With fewer historical paths, the confidences generated may not represent the actual movement patterns of the intruder. Hence, the chance of missing the intruder is high. As the number of historical path increases, the confidences

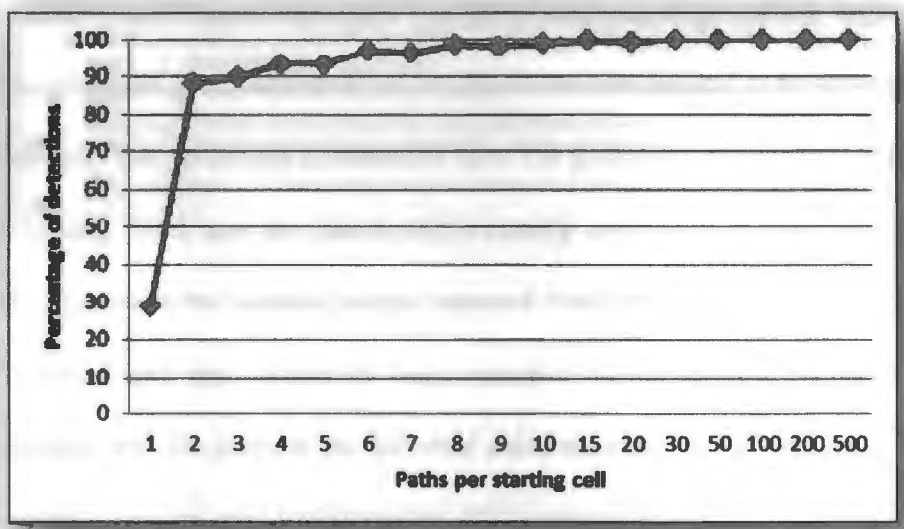


Chart 11.14. Effect of Varying the Number of Historical Paths on the Detection of Intruder When Using TABA

generated represent the actual trend of the intruder's movement pattern and consequently the chance of correctly detecting the intruder increases.

12. CONCLUSION

In this thesis we have presented three sensor scheduling and intruder tracking algorithms; the Tactic Association Based Algorithm, the Tactic Case Based Algorithm, and the Tactic Weight Based Algorithm. The **TABA** uses sequence pattern mining to generate confidences of movement of an intruder from one location to another in the sensor network. The generated confidences form the basis for sensor scheduling and intruder tracking. **TCBA** uses the case-based reasoning approach to schedule sensors and track intruders in the wireless sensor network. The **TCBA** compares the current intuder location and the number of steps required by the intruder to reach the current location with the paths in the historical paths database to find simmilar cases. The generated cases are used for scheduling the sensors and tracking the intruder in the sensor network. In the **TWBA** we use the hexagonal representation of the surveillance area. Each hexagon in the grid is assigned a weight. The weight represents the complexity of the terrain that the hexagon encompasses. The weight is used to generate movement probabilities from each hexagon in the grid. The movement probability includes the probability of movement of an intruder from the current hexagon to all the adjacent hexagons. The movement probabilities are the basis for scheduling in **TWBA**.

In this research we also introduce a novel approach to generate probable intruder paths which are strong representatives of the paths intruder would take when moving through the sensor network. We use the concept of terrain complexity to generate probable intruder paths. The intruder is assumed to be an intelligent entity; he is aware of the terrain complexity around him and tries to find a path to a selected destination through a least complex terrain. The representative paths generated by using this method can be used for planning and designing of the sensor network and also in sensor scheduling.

We developed the Tactic Evaluation and Simulation Tool (**TEST**) to effectively analyze and simulate the performance of the tactics. The **TEST** enables visualization of all the sensor scheduling and intruder tracking algorithms and the communication algorithm. The simulator consists of two panes; the left pane is used to draw the simulation and the right pane consists of controls used to regulate the simulation. The tool has been created using C Sharp.

Experimentation was performed to determine the performance of each of the sensor scheduling and tracking algorithm. The experimental results show that all the three sensor scheduling and intruder tracking algorithms perform well in tracking an intruder. The dynamic decision making enables selective sensor scheduling which helps in conserving sensor battery power. The algorithms have also been shown to be fault tolerant. The algorithms adjust to the changing topology of the sensor network as the sensors die without compromising the precision of intruder tracking. The algorithms have also shown to be effective in tracking multiple intruders simultaneously. The algorithms are scalable and are shown to work effectively for a 10*10 and 50*50 sensor network grid.

Comparisons between the three algorithms show that life of a sensor network using **TABA** is greater than **TCBA** and **TWBA** algorithms. The life of a sensor network using **TCBA** is the least. The **TABA** can handle multiple intruders more efficiently than **TCBA** and **TWBA**. **TWBA** has the highest accuracy in predicting the intruder position followed by **TABA** and least accurately by **TCBA**. TWBA also performs the best when there are many sensors in the sensor network.

13. FUTURE WORK

The three algorithms discussed in this thesis have been programmed and tested to work independently. In future, we plan to develop a scenario driven deployment of the scheduling and tracking techniques. We also plan to research the affectivity of interleaved operation of the scheduling and tracking algorithms.

We plan to develop a concrete recovery algorithm and incorporate it with the sensor scheduling and tracing algorithms. The job of the recovery algorithm would be to track down an intruder when the sensor scheduling and tracking algorithm loses track of the intruder.

The scheduling and tracking algorithms have been developed to work with homogeneous sensor networks. We plan to research the application of these algorithms in the heterogeneous sensor networks with mobile sensors.

14. REFERENCES

1. *Wireless sensor networks: a survey.* **I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci.** 2002, Computer Networks

2. *Dual Prediction-Based Reporting for Object Tracking Sensor Networks.* **Yingqi Xu, Julian Winter, Wang-Chien Lee.** 2004, First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)

3. *A survey on sensor network.* **Akyildiz, I. F. Su, W. Sankarasubramaniam, Y. Cayirci, E.** s.l. : IEEE Institute of electrical and electronics, 2002, IEEE Communications magzine, Vol.40

4. *Survey on wireless sensor network applications.* **Mao Xiaofeng, Yang Min, Mao Dilin.** s.l. : Computer Applications and Software, 2008 , Vol. 25,.

5. *A survey of energy-efficient scheduling mechanisms in sensor networks.* **Lan Wang, Yang Xiao.** Volume 11, Number 5 / October, 2006, s.l. : Springer Netherlands, October 2006, Mobile Networks and Applications .

6. *A mac protocol to reduce sensor network.* **M. J. Miller and N. H. Vaidya.** May/June 2005, IEEE Transactions on Mobile Computing, Vol. 4(3).

7. *Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks.* **E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, A. Chandrakasan.** 2001, Proceedings of ACM Mobicom '01.

8. *Protocols and architectures for wireless sensor networks.* **Holger Karl, Andreas Willig,** June 2005, Wiley publications.

9. *Energy efficient target tracking in a sensor network using non-myopic sensor scheduling.* **Chhetri, A.S. Morrell, D. Papandreou-Suppappola.** 2005. Proceedings of 8th International Conference on Information Fusion.

10. *Achieving real-time target tracking using wireless sensor networks.* **Tian He Vicaire, P. Ting Yan Liqian Luo Lin Gu Gang Zhou Stoleru, R. Qing Cao Stankovic, J.A. Abdelzaher, T.** 2006. Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE .

11. *A coverage-preserving node scheduling scheme for large wireless sensor networks.* **Georganas, D. Tian and N. D.** 2002. Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02).