UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Location Inaccuracies in WSAN Placement Algorithms

By

Gareth Michael Nicholls

February 2010

Submitted in partial fulfilment of the requirements for the degree Master of Science (Computer Science) in the Faculty of Engineering, Built Environment and Information Technology, University of Pretoria, Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

I declare that the thesis that I hereby submit for the degree MSc Computer Science at the University of Pretoria is my own work and has not previously been submitted by me for degree purposes at any other university or institution.

______________________________ ___________________
DATE

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

Wisdom, Strngth and Beauty

ABSTRACT

The random deployment of Wireless Sensor and Actuator Network (WSAN) nodes in areas often inaccessible, results in so-called coverage holes – i.e. areas in the network that are not adequately covered by nodes to suit the requirements of the network. Various coverage protocol algorithms have been designed to reduce or eliminate coverage holes within WSANs by indicating how to move the nodes. The effectiveness of such coverage protocols could be jeopardised by inaccuracy in the initial node location data that is broadcast by the respective nodes. This study examines the effects of location inaccuracies on five sensor deployment and reconfiguration algorithms – They include two algorithms which assume that mobile nodes are deployed (referred to as the VEC and VOR algorithms); two that assume static nodes are deployed (referred to as the CNPSS and OGDC algorithms); and a single algorithm (based on a bidding protocol) that assumes a hybrid scenario in which both static and mobile nodes are deployed. Two variations of this latter algorithm are studied.

A location simulation tool was built using the GE Smallworld GIS application and the Magik programming language. The simulation results are based on three above-mentioned deployment scenarios; mobile, hybrid and static.

The simulation results suggest the VOR algorithm is reasonably robust if the location inaccuracies are somewhat lower than the sensing distance and also if a high degree of inaccuracy is limited to a relatively small percentage of the nodes. The VEC algorithm is considerably less robust, but prevents nodes from drifting beyond the boundaries in the case of large inaccuracies. The bidding protocol used by the hybrid algorithm appears to be robust only when the static nodes are accurate and there is a low degree of inaccuracy within the mobile nodes. Finally the static algorithms are shown to be the most robust; the CPNSS algorithm appears to be immune to location inaccuracies whilst the OGDC algorithm was shown to reduce the number of active nodes in the network to a better extent than that of the CPNSS algorithm.

Keywords: Geo-information systems, coverage hole, Voronoi polygons, Delaunay triangle, mote, node

Degree: Magister Scientia
Supervisor: Prof D.G. Kourie
Co-Supervisor: T. Strauss

ACKNOWLEDGEMENTS

My thanks to the following people who helped me directly, inspired me or simply encouraged me during the writing of this study. Derrick Kourie and Tinus Strauss for the countless reviews and re-reviews, help, comments and general commitment to helping me finish. My family, my Mom and Dad, Jason Nicholls. Christiaan Marais for his statistical help. Hardus du Toit and Herman Smith for the Magik development assistance. To GENS Smallworld, PowerTech IST and Paresh Joshi for the introduction to Smallworld and use of development licenses.

It goes without saying, I hope, none of these people are responsible for any errors or inconsistencies…

Table of Contents

2 List of Figures

3 List of Tables

Chapter 1

Introduction

What we call the beginning
is often the end.
And to make an end is
to make a beginning.
The end is where we start from.

T.S. Eliot, Four Quartets, 1943

A Lockheed C-5 cargo plane[1] *flies over Robben Island*[2] *at an altitude of 1000 feet and a speed of 600km/h. The cargo hold opens up and thousands of small sensory devices, approximately 20 per cubic meter are deployed. As the devices come to a standstill on an open field they broadcast a signal to all listening nodes. The signal contains their current location, deployment status and signal strength. A small percentage of the nodes broadcast to the rest a message that their coverage area does contain enough nodes to be adequately monitored. On the other side of the field a hand full of self-organising devices, devices that are capable of movement, respond back and move into a position to assist the requesting nodes. The purpose of these devices is to sense and track penguin movement on the island.*

1.1 Introduction

Wireless Sensor and Actuator Networks (WSANs) have seen a growth in research within the past few years. Research interests include hardware development to reduce the manufacturing costs, software development in the form of network management interfaces systems (NMIS), network configuration and routing algorithms. The deployment of these networks into the real-world is prone to failure of nodes within the network, or unpredictable network layout. The following study attempts to evaluate so-called deployment and reconfiguration algorithms, or simply coverage algorithms. The purpose of these algorithms is to solve coverage holes within the network topology, where coverage holes are defined to be areas in the network that are not adequately covered by sensors.

[1] Commonly known as the Lockheed Galaxy, is a cargo plane used by the USAF to transport military cargo and supplies (http://www.lockheedmartin.com/products/c5/).
[2] Robben Island is a small island off the West Coast of Cape Town, South Africa. The island, which directly translated is Seal Island in Dutch, was used as a prison between 1961 and 1991. The island is now owned by the state and is a popular tourist attraction due to its political past and current abundance of wildlife (ICOMOS 1999)

Meguerdichian *et al.* (2001a) present a study focussed on the coverage problems presented in Wireless Sensor Networks. The study divides networks into two key categories, namely *deterministic coverage* and *stochastic coverage*. They also present various ways of determining coverage holes within the network. One such algorithm is that of using *Voronoi polygons*, a key component in computational geometry. The components making up the studies by Meguerdichian *et al.* (2001 a, b) form key aspects of this study and are discussed in detail further on.

1.2 Need for Research

As we will present in the following study, there exists many algorithms to manage *post deployment* of networks, reducing node redundancy and coverage holes. However almost all of the algorithms make the assumption of accurate location information provided to the nodes in the network. This factor, as far as can be determined, is never taken into account by the original authors of the algorithms. This study attempts to analyse the effects location have on the deployment and reconfiguration in a mock environment simulated with the aid of GIS systems.

1.3 Approach

For the purposes of examining the extent to which the deployment and reconfiguration algorithms are capable of reducing coverage holes, a simulator environment was set up, using a custom-designed simulation tool. Six simulation experiments are performed on each of the reconfiguration algorithms. The deployment setup and environment are kept constant to allow for equal comparison between algorithms and experiments.

In the study presented, we are particularly interested in coverage in terms of overall sensing capability. It is assumed that the extent of coverage in terms of inter-node communication signals is sufficient for all the nodes to communicate to the base station via the network. During this and further chapters the terms nodes, sensors, devices and actuators are used interchangeably

1.4 Research Questions

Chapter 2 examines the relationship between location and coverage algorithms. The estimated location of the nodes in a real-world environment cannot hold to the assumption that the location will always be accurate. This study evaluates five (with a sixth being a variation of assessment) deployment and reconfiguration algorithms based on levels of location inaccuracy. The following questions are proposed:

- What are the effects of location inaccuracy on deployment and reconfiguration algorithms used in WSANs?
- How do these results compare between mobile, hybrid and static networks?

1.5 The Way Ahead

Chapter 2 below introduces WSANs and the need for coverage reconfiguration and deployment algorithms. We show the main assumption of the algorithms to be that of location awareness amongst the nodes and hence the need to evaluate the effects of location inaccuracies on the algorithms. The chapter shows how location inaccuracies may typically be presented in a real-world context. Chapter 3 proposes an architecture for a GIS based simulator. By integrating a simulation tool with that of a GIS platform, we are able to place nodes within a real-world environment. The GIS environment also provides an industry tested set of mathematical libraries that may assist in the assessment of the algorithms. Chapter 4 then describes each of the five algorithms evaluated. The chapter also examines the methods and procedures used for evaluation. The chapter closes by examining the initial deployment of nodes as well as the introduction of location inaccuracies. The results of the simulator are presented in Chapter 5 to indicate each of the algorithms' effectiveness in comparison to the experiments. Finally in Chapter 6 a conclusion is drawn from the results found in Chapter 5. In this chapter we present further areas of research for future study as well as current related topics.

Chapter 2

Background Information-WSAN

"Everything is related to everything else,
but nearby things are more
related than distant things"

W. Tobler, 1970 –
1st law of Geography

This Chapter outlines the concept of Wireless Sensor and Actuator Networks (WSAN); it builds a foundation for further Chapters, discussions and findings.

Section 2.1 discusses the concepts of WSAN design and deployment; Sections 2.2 describes how these networks are deployed with a special emphasis on the types of networks that can be deployed. The chapter then introduces the concepts of coverage, coverage holes and Voronoi -polygons. We then see how these concepts are tightly coupled to an assumption of location information being provided to the nodes within the network.

2.1 Wireless Sensor and Actuator Networks

Recent developments in micro-electronic mechanical systems (MEMS) including the development of wireless transfer mediums and micro-robotics have seen a growth in the field of WSAN research. These networks consist of one or more base stations, Figure 2-1, and tiny nodes or motes (potentially thousands of them), Figure 2-2, which are scattered in a given region of interest (ROI), to sense and monitor the surroundings (Alkyildiz *et al.* 2002).

Figure 2-1 - MicroStrain Base station.

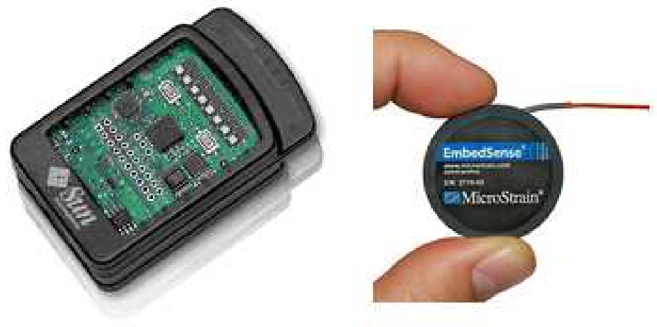


Figure 2-2 - Two motes, by Sun Microsystems (Sunspots) and MicroStrain.

The nodes have hardware onboard capable of sensing seismic activity, temperature, acoustics, and visual objects. If need be almost any other form of sensing can be installed. (Alkyildiz *et al.* 2002, Xbow, Java SunSpots, MicroStrain) These networks are different from the conventional mobile ad-hoc networks because of their ability to sense data, to filter data at the node level, and to relay the processed data to base stations via the network for further processing (Alkyildiz *et al.* 2002). By passing the data along neighbouring nodes, a global view of the area is obtained by the network. The base stations are components within the WSAN that are able to contain greater computational, energy and communication abilities. The nodes may also be equipped with actuators that allow them to react and perhaps change their environment, based on messages received from the base station.

Alkyildiz *et al.* (2002); Ahmed *et al.* (2005) and Santi (2005) were used to compile the following list of aspects that should be taken into account before the sensors are deployed over a ROI.

Energy conservation – Unlike wired networks, wireless sensor networks have a limited energy supply. Once the nodes are deployed, replacement or recharging of the energy source in the ROI is sometimes impractical or impossible. Thus, a key goal in WSAN is to reduce the energy consumption of nodes within the network.

Limited bandwidth – The most commonly used communication standard in WSAN is that of IEEE 802.11. The theoretical threshold of this standard is 54Mb/sec, however in real-world deployment, simultaneous communications increases radio interference. The degree of interference depends on the density of deployment, i.e. the higher the concentration of nodes that are deployed over the ROI, the higher the degree of radio interference.

Unstructured and varying topologies – When the nodes are deployed from a given source, for example the C-5 aircraft, the final layout of the network is unpredictable. Node failure during deployment as well as during implementation changes the layout of the network dynamically.

Fault tolerance – Fault tolerance is linked to varying topologies. During the deployment phase or lifetime of the network, nodes may malfunction or run low on energy. This failure of nodes, in theory, should not affect the overall purpose of the network.

Low quality in communication – Radio interference as well as different environmental medium such as water, soil and vegetation reduce the quality of the signal strength.

Data processing – The processing power within the network is limited. When deployment of a WSAN is done, communication of data between nodes should be considered. This aspect is tightly coupled with that of communication quality and low bandwidth. The nodes should be able to determine what sensed data should be passed on to neighbouring nodes.

Scalability – A final detail that should be considered is that of scalability. WSANs can comprise of thousands of nodes. Scalability is concerned with how the network will handle growth in workload or support an increase in the number of nodes.

Meguerdichian *et al.* (2001a) present two forms of deployment; the first is that of deterministic coverage which is a network that has been deployed to a specific shape known by the network designers. Deterministic networks provide an ideal network where the designers have prior knowledge of the position of nodes within the network. Secondly the study presents stochastic coverage. In many of the potential working environments[3] such as toxic disaster areas, remote harsh fields or even remote planetary exploration, sensor deployment cannot be performed manually. In the example for nature observation on Robben Island, an aerial deployment source is used to deploy the sensors. However, by using techniques such as this, the final location of the nodes cannot be predicted and deemed as optimal. Furthermore malfunction of nodes' results in further network layout changes. Thus, it can be said the layout of the network is unpredictable. Such random deployments will invariably

[3] BP is currently implementing WSN to transform business processes, from inventory control to monitoring pumps (CompuWorld 2005)

result in the occurrence of coverage holes within the ROI. Ahmed *et al.* (2005) describe the notion of a coverage hole (or rather the absence thereof) as follows:

> *"Given a set of sensors and a target area, no coverage hole exists in the target area, if every point in that area is covered by at least k sensors, where k is the required degree of coverage for a particular application."*

Meguerdichian *et al.* (2001 a) make the claim that coverage can be considered as the measure of *quality of service* of the sensor network. Ahmed *et al.* (2005) confirm this claim: the service provided by the sensor network is dependent on the ability of the sensors to monitor the area adequately, based on the degree of coverage required. As with the example used before, the nodes on Robben Island reported that areas within the network did not meet the above criteria.

The inverse of coverage holes also applies. This takes the form of *over-coverage*; these are areas in the network that waste nodes sensing ability by duplicating coverage in specific areas.

A distinction should be made between the extent to which a node is covered with respect to communication signals that are sent between nodes; and the extent to which sensing capability that radiates from the various nodes cover the ROI. The extent of coverage will typically differ from one case to the other.

Various nodes within the network could be referred to as *movement-assisted*. By movement-assisted nodes, we refer to nodes that are able to move within the ROI via some agent. We refer to research such as (Khan *et al.* 1999, Heo *et al.* 2003, Wang *et al.* 2003a, Wang *et al.* 2003b) for active movement-assisted networks. The researchers discuss nodes that are capable of self-movement as well as nodes that use an external source for movement; such a source could be an unmanned aerial vehicle or an autonomous robot.

The deployment of WSAN can be classified in three distinct groups based on the percentage of movement-assisted or self-organising nodes within the network: (i) Mobile networks; (ii) Static networks; and (iii) Hybrid networks.

To solve problems of coverage holes, various self-organising or coverage algorithms have been developed to allow the networks to reconfigure their network topologies based on the requirements of the application. The reconfiguration algorithms give the network the ability to reduce coverage holes in much the same way as described in the penguin observation example used earlier.

2.2 Deployment and Reconfiguration

Several researchers (Wang *et al.* 2003a, Wang *et al.* 2003b, 2004; Cortes *et al.* 2002; Generiwal *et al.* 2004; Huang *et al.* 2003; Heo *et al.* 2003) have presented solutions to obtain the required degree of coverage over the network by reconfiguring the nodes position. A generalised problem statement is to minimise the coverage holes or over-coverage with constraints on deployment and reconfiguration time as well as energy used by each node (Wang *et al.* 2004, Ahmed *et al.* 2005). This optimisation of placement of nodes within the ROI can be classified as a spatial resource allocation problem. The study of this optimal placement is the subject of a discipline called *location optimisation* (Cortes *et al.* 2002). Location optimisation is a broad field of study ranging from the study of WSAN networks to the study of animal territorial behaviour (Cortes *et al.* 2002, Righton 2006). In the case of stochastic deployment, or deployment that is random, optimal placement is done by reconfiguring and reorganising the network.

2.2.1 Mobile Sensor Networks

Mobile sensor networks are networks that are deployed with all the nodes being movement-assisted. Thus, each node is able to move and reconfigure its location. A typical problem statement for mobile networks is to minimise coverage holes with the same constraints on deployment and reconfiguration time and energy as well as the distance moved by each node (Ahmed *et al.* 2005).

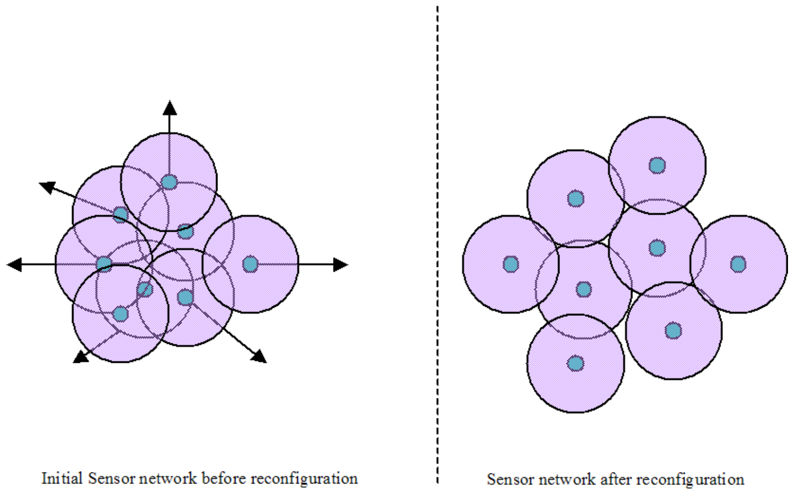

Initial Sensor network before reconfiguration
Sensor network after reconfiguration

Figure 2-3 - Reconfiguration of a mobile network.

Figure 2-3 represents a hypothetical reconfiguration of a mobile network. Within a densely populated area in the network, reconfiguration takes place in the form of nodes moving away from each other, spreading themselves out over a greater area or moving into areas that are not adequately covered. The nodes within the network relocate to a location that will allow the node to optimally cover the ROI.

The current study focuses on two movement-assisted reconfiguration algorithms implemented by Wang *et al.* (2004). The algorithms detect the presence of coverage holes via the aid of Voronoi polygons. After the detection of a coverage hole, the purpose of the protocols is to calculate a new target location for the node, where the target location is defined as the position where the node can cover its local area optimally.

2.2.2 Hybrid Sensor Networks

Due to the energy requirements and manufacturing costs of movement-assisted sensors, a proposed solution is to limit the number of mobile nodes. The movement-assisted nodes are able to assist in deployment and network repair by moving to appropriate locations within the network topology.

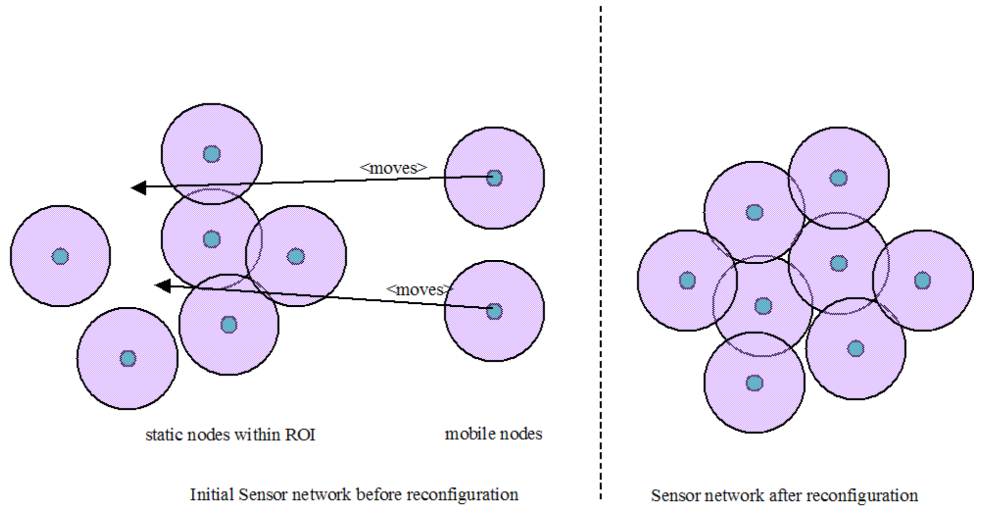

<moves>
<moves>
static nodes within ROI
mobile nodes
Initial Sensor network before reconfiguration
Sensor network after reconfiguration

Figure 2-4 - Reconfiguration of a hybrid network.

Figure 2-4 shows the movement of two mobile sensors into areas within the ROI. The static sensors calculate the presence of coverage holes and determine the locations to which the mobile nodes should move.

Two variations of a hybrid sensor network algorithm are assessed in the current study. The algorithm, presented by Wang *et al.* (2003a) is an extension to their movement-assisted algorithms presented in Wang *et al.* (2004). The algorithm, which uses a multiple mobile sensor approach (discussed further in Chapter 4), allows static sensors to bid for the usage of the many mobile sensors to solve the static nodes' local coverage holes.

The first variation of the algorithm is assessed from the perspective of the static nodes being inaccurate. By this we mean that the calculations of location by the static nodes are deemed to be vulnerable to inaccuracy. The second approach is to assess the algorithm from the perspective of the mobile nodes. In this case we deem the mobile nodes to be inaccurate.

2.2.3 Static Sensor Networks

Due to the nodes within a static network being fixed to their initial location during the lifetime of the network, the static nodes are limited in their ability to reconfigure the networks optimally i.e. to repair network configuration. Reconfiguration of static networks are concerned with topology /density control. By topology we refer to the way the network is configured based on location i.e. the way the network's physical layout. Density control is the reference made to the concentration of nodes within a given area or across the entire ROI. Figure 2-5 shows how a static network is '*reconfigured*' by switching off certain nodes within the network or increasing the sensing radius of other nodes to provide coverage. The nodes are aware of the required coverage needed by the network. The configuration algorithms determine which nodes within the network should reduce their signal strength or even switch off temporarily to reduce signal overlapping. This process of switching nodes on and off is referred to as *node scheduling* (Tian *et al.* 2002, Ruzzelli *et al.* 2005).

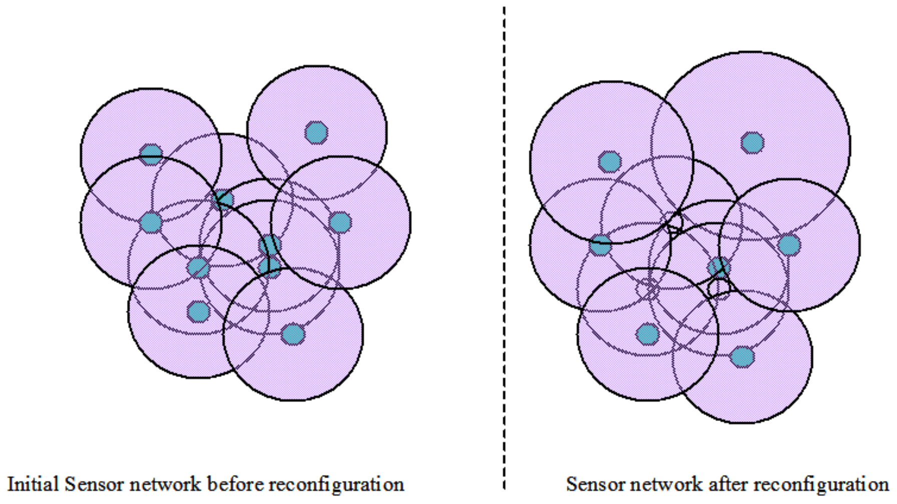

Initial Sensor network before reconfiguration
Sensor network after reconfiguration

Figure 2-5 - Reconfiguration of a static network.

As in the case of mobile and hybrid networks, two static reconfiguration algorithms were chosen. These static algorithms are tightly coupled to one another. Tian (2002) presents the first. The algorithm is an extension on the *Low-energy adaptive clustering hierarchy* (LEACH) algorithm (Alkyildiz *et al.* 2002). The LEACH algorithm uses clustering to form groups of nodes with a single cluster head. All sensors within the group, excluding the head, power down and wait for the head node to notify the *sleeping* nodes. Randomisation is used to rotate the responsibilities of the cluster head amongst nodes within the group. Groups are determined based on the sharing of coverage areas. The algorithm uses this approach of node scheduling; however nodes decide to turn on or off after discovering neighbouring nodes that can completely cover its sensing area. The sensing area for the neighbouring nodes is determined as sectors within its coverage area (Alkyildiz *et al.* 2002).

The second static algorithm is much the same as the above, implementing a combination of node clustering and scheduling. The *Optimal Geographical Density*

Control Algorithm (OGDC) as presented by Zhang and Hou (2003) is an iterative algorithm that divides the network lifetime into timed rounds. At each round the network is 'woken up' after which the algorithm is run to determine which nodes are to hibernate until the next round.

All five of the above-mentioned algorithms are discussed in more detail within Chapter 4.

2.3 Location Dependency

Table 1 represents a comparison of the various self-organising algorithms mention in the previous section. Presented by Ahmed *et al.* (2005), the table outlines the main assumptions and characteristics of each algorithm. A reoccurring assumption is that nodes are location aware. However, in practice, location calculation or estimates are not very accurate. Oliveira *et al.* (2005 a) identify this as the localization problem.

Table 2-1 - Comparison of proposed solutions to coverage hole problem.

Category	*Approach*	*Proposed Solution*	*Main Assumptions*	*Characteristics*
Mobile Sensors	Computational Geometry	VEC, VOR, Minmax	Location information	Localised, scalable, distributed.
		Co-Fi	Location information, nodes can predict their death	Single coverage based. Residual energy considerations.
	Virtual Forces	Potential Fields	Range and bearing	Scalable, distributed. No local communication required for localisation
		DSS, IDCA	Location information	Scalable, distributed, residual energy based.
	Sequential	Incremental	Line of sight for localisation	Centralised. Bidirectional communication with base station.
Hybrid Sensors	Single Mobile Sensor	UAV	Predetermined topology information	Flying robot for deployment and network repair. Inaccuracies using aerial deployment.
		Single Robot	Location information	Distributed, no multi-hop communication for network deployment and repair.
	Multiple Mobile Sensors	Bidding Protocol	Location information	Uses Voronoi-diagram for single coverage requirement.

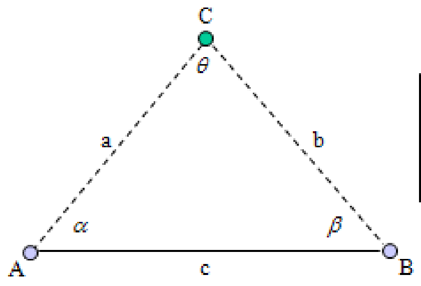
Category	Approach	Proposed Solution	Main Assumptions	Characteristics
Static Sensors	Multiple Coverage	CCP	Location information, uniform sensing disk	Configurable degree of coverage, calculated by intersection points of sensing circles.
		k-UC, *k*-NC	Location information	Perimeter coverage, non-disk sensing model supported.
		Differentiated	Location information, time synchronisation	Grid based differentiated degree of coverage.
	Single Coverage	OGDC	Location information, uniform sensing disk	Residual energy consideration.
		Sponsored Area	Location information	Sector based coverage calculations, non-disk sensing model supported.
		Extended-Sponsored Area	Location information, time synchronisation	Uniform disk sensing model.

As the layout of the network is not predefined, calculating the location of each node is required. An optimal approach to determine accurately the location of a node would be to use *geo-location* by fitting a *Global Positioning System* (GPS) to each node in the network. However this solution adds to the manufacturing cost, as well as to the energy consumption of each node, and this is generally not considered practical. The usage of GPS is also limited by the environmental conditions, i.e. the availability of a GPS signal which is dependent on the location of the nodes within a building, presence of vegetation, etc. all of which introduce a degree of inaccuracy (Kennedy 1996). Thus, normally in a real-life situation, some location algorithm is used to determine the location of nodes within the network. Triangulation and radio-location are two basic mathematical approaches used in location calculation algorithms. The algorithms will accurately calculate location in a 2-D plane. However, in applying these algorithms, in a real-world placement, with Earth contours and obstacles, inaccuracies in the calculation of locations are likely to occur.

For completion we discuss the principles of triangulation and radio-location next. Triangulation is a well-known process of calculating the position of a third location (C) when two vertices (A, B) are already known. Using the properties of triangles and the *laws of sine*, the position of a third point, a vertex, can be calculated. The sine law states that:

"The sides of a triangle are to one another in the same ratio as the sines of their opposite angles."

Equation 2-1



$$\text{In } \triangle ABC, \quad \frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c}$$

This statement shown in Equation 2-1 means that the angle between each of the known vertices and the third vertex is measured, the angle of the third vertex (θ) can be determined as 180° less the angle at A(α) and B(β). The distance, *c*, from A to B is known via the co-ordinates of A and B. Using the ratio *sin θ/c*, the length of AC and BC can be calculated, and from this, the co-ordinates of C are easily determined using a Cartesian plane to represent x and y co-ordinates. Figure 2-6 illustrates the general idea.

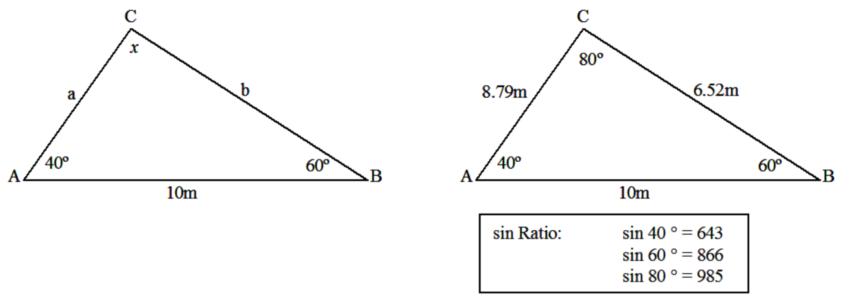


Figure 2-6 - Using sin to calculate the length of *a, b*

The value *x* at vertex C can be calculated as *x* = 180° - 60° - 40°, resulting in *x* = 80°. The laws of sine say that the ratio of each line opposite the angle is calculated as sinθ, where θ is the vertex degree opposite the line. Using this claim the sides are now in the ratio, 643 : 866 : 985. Using Equation 2-1, it can be said that

$$\frac{\sin 40°}{a} = \frac{\sin 60°}{b} = \frac{\sin 80°}{10}$$

Therefore: $a = 10\text{m}.\dfrac{866}{985} = 8.79\text{m},$

and $b = 10\text{m}.\dfrac{643}{985} = 6.52\text{m}$

In the context of WSAN, the concept of *mesh triangulation* is used to calculate the co-ordinates of nodes within the network. Mesh triangulation is the process of using more than two neighbours to confirm the location of a point. This is done by using triangulation for pairs of neighbours to that point, each calculating the location of the point. Once each pair has a value, the mean position of all the neighbours' calculated values are set to be the location of that point. Niculescu (2003) shows that the positions of nodes that do not have a sufficient number of neighbours to calculate their present location, such as those at the ROI boundary, tend to be less accurate than those with an abundance neighbours, i.e. at the centre of the ROI.

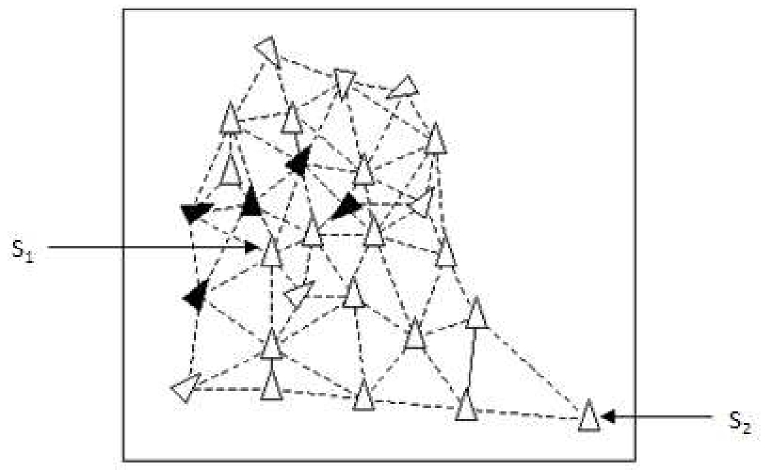


Figure 2-7 - Calculation of location by radiolocation / triangulation.

Figure 2-7 depicts a typical scenario using mesh triangulation. The darker triangles represent nodes in the network with GPS capability. Pairs of GPS enabled nodes calculate the location of a third neighbouring node. Each node that is aware of its location then calculates the location of another neighbouring node. As can be seen in the above figure, nodes with neighbours in close proximity, S_1, are more likely to have accurate locations due to the confirmation and recalculation of locations by more neighbours at different locations than nodes with only two neighbouring nodes, S_2.

Based on the same principles as triangulation, radio-location is the process of finding the location of a node by means of radio waves. Radio signals are sent by node S_i to the neighbouring nodes. The angle (AOA – Angle of Arrival) at which the signal returns to S_i, as well as the time (TOA – Time of Arrival) it takes to return are used to calculate the location of the neighbouring nodes.

Using triangulation and radio-location, researchers have developed location calculation algorithms or localization algorithms, such as the ad-hoc positioning system (Niculescu 2003), recursive position estimation (Albowicz *et al.* 2001) and direct position estimation (Oliveira *et al.* 2005a) to estimate positions of objects within a wireless network. Oliveira *et al.* (2005a) discuss three key components in determining the overall location of a node: *distance estimation*, which is responsible for estimating the distance between two objects, using either radio-location or triangulation; *position computation*, which is the component that calculates the position of an object with the aid of current information such as neighbouring objects or distance to neighbours, as in the case of mesh triangulation; and finally the *localization algorithm*, which is responsible for using the above two criteria and other information in the calculation of location.

Oliveira *et al.* (2005a) present a study on the dependency of each of these criteria and how they affect the overall outcome of location calculation. The study also presents degrees of error with the calculation, showing that the calculation of a node's location within a wireless sensor network cannot be deemed to be completely accurate. Other researches such as Simon *et al.* (2004) present inaccuracies within the location calculation. However they deem the algorithm's degree of error to be acceptable for the needs of their applications.

Potential problems arising from location inaccuracies are the inaccurate calculation of coverage holes as well as optimal positions for nodes to adjust. The accurate calculation of *Voronoi polygons* is dependent on the location information provided. This claim forms a key component of this study, where the study aims at assessing the impact that inaccuracies will have on the coverage algorithms.

2.4 Voronoi polygons

The construction of a Voronoi-diagram[4] is based on Tobler's first law of Geography (Sharifzadeh 2004), which states that:

> *"Everything is related to everything else, but nearby things are more related than distant things"*

The location information about proximity to neighbours allows for the construction of a Voronoi proximity polygon (or simply Voronoi polygon) around

[4] Voronoi-diagrams are often referred to as Voronoi tessellation, Voronoi decomposition or Dirichlet tessellation (Aurenhammer 1991).

each node (Aurenhammer 1991). Figure 2-8 shows an example of a Voronoi-diagram for a set of randomly places points. Before explaining how to construct such a diagram, an indication is given of how these polygons are used to determine the size of the coverage holes in a ROI.

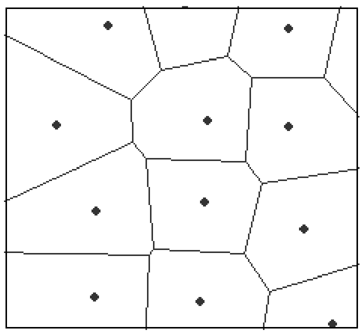


Figure 2-8 - Voronoi-diagram for a randomly placed set of points on a 2-dimensional plane

2.4.1 Voronoi polygons and the Relationship to Coverage Holes

The set of Voronoi polygons around all nodes in the ROI constitute a complete partition of the ROI. In making this claim, it is assumed that the ROI is bounded by a convex polygon, and using that boundary, the Voronoi polygons are appropriately adapted so that their sides coincide with those of the ROI.

Each Voronoi polygon indicates a local area of coverage for which a node should be made responsible. The area of such a Voronoi polygon that falls outside the circle of coverage of its associated node (as shown in Figure 2-9), can be used to determine the overall size of the holes in the network.

The total coverage hole can be determined as a percentage within the ROI by Equation 2-2. The equation shows that the total coverage hole, H, is equal to the area of the ROI less the union of all the sensors (S) coverage area, where coverage area of sensor S_i is deemed to be the coverage included within the ROI, i.e. excluding is any coverage that overlaps outside the ROI.

Equation 2-2

$$\boxed{H = ROI^2 - \bigcup_{i=1}^{n} A(Si)}$$

An example is given in Figure 2-9. The figure shows a WSAN network over a given ROI. In this particular case, there are not enough nodes to cover the entire ROI, so coverage holes are inevitable.

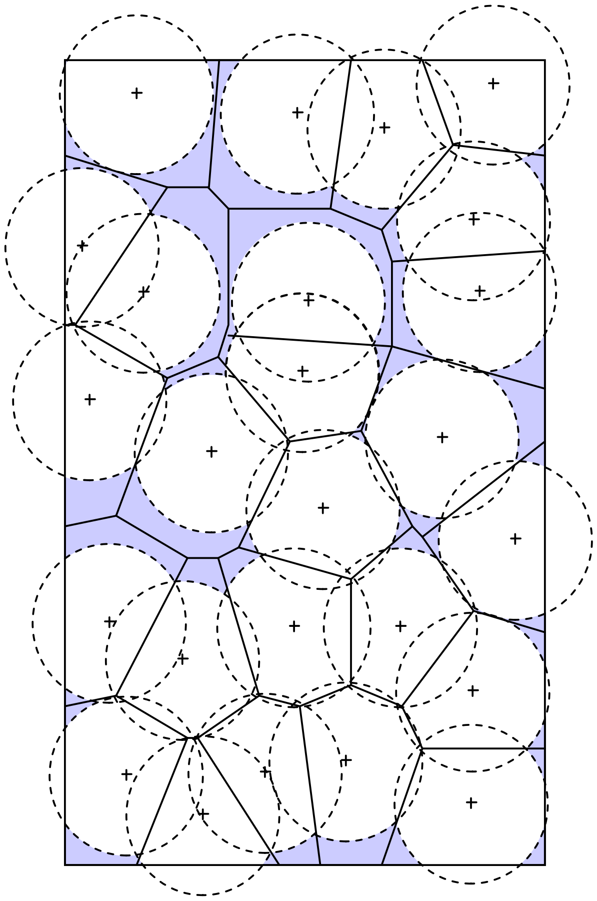


Figure 2-9 - Determining the existence of a coverage hole within to ROI.

Within Figure 2-9 the coverage hole areas are represented as the darker regions. The figure shows a Voronoi-diagram for a specific ROI containing 25 nodes. The nodes are represented by the + symbol, with the sensor coverage represented by the area of the circle in white. Using the algorithm above, the percentage of darker regions in relation to the ROI can be calculated. In this particular case the total coverage hole percentage was calculated as 14.12%.

2.4.2 Construction of Voronoi polygons

A Voronoi polygon of a node has the property that each point in it is closer to its associated node than to any other node in its surroundings. A Voronoi-diagram, a decomposition of the ROI, is the result of determining all the Voronoi polygons in the

ROI. Voronoi polygons constitute important objects in computational geometry and GIS-based applications (Aurenhammer 1991).

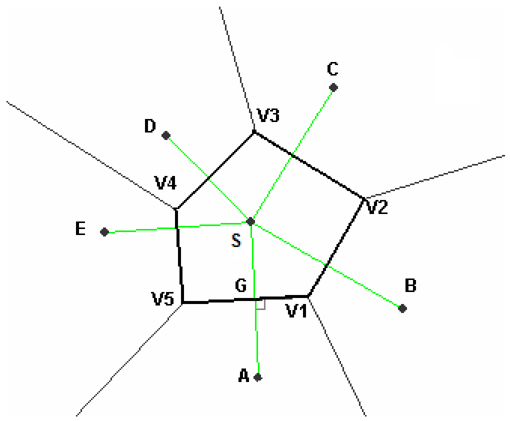


Figure 2-10 - Voronoi polygon Gp(S) of point (S) (Wang 2004).

Figure 2-10 represents a Voronoi polygon of some node, S. The Voronoi polygon of S, can be represented by (V_p(S), E_p(S)), where V_p(S) is the set of Voronoi vertices, E_p(S) is the set of Voronoi edges. Thus, in the example given in Figure 2-10 (Wang *et al.* 2004), V_p(S) = {v1, v2, v3, v4, v5}, E_p(S) = {v1 v2, v2 v3, v3 v4, v4 v5, v5 v1}. Furthermore, the set of neighbouring nodes of S is {A,B,C,D,E}.

Each line drawn between node S and a neighbouring node in this set will intersect an edge of the Voronoi polygon at right angles. This line will also be bisected by this polygon edge. Thus, line (A,S) intersects (v5,v1) at right angles, and is bisected by (v5,v1). These direct neighbours are deemed as the closest points to S, where closest is defined as straight-line distance.

Various algorithms (Aurenhammer 1991) have been created for accurately and efficiently calculating the Voronoi polygons and consequently the Voronoi-diagram. The following section describes in the form of pseudo code, a high level view of the process of creating a Voronoi-diagram given, a set of nodes.

Table 2-2 - Pseudo code for the construction of a Voronoi-diagram.

```
1. For each node, $S_i$, in the ROI, generate a Voronoi polygon as follows:
    a. Find $S_j$, the closest node to the $S_i$
    b. Draw a line bisecting the line connecting $S_i$ and $S_j$
    c. Select the next closest neighbour $S_k$ to $S_i$
    d. Draw a line bisecting the position of $S_k$ and $S_i$
        i. If two lines intersect, the meeting point is a vertex of the
           polygon
        e.   Repeat until a closed polygon is created around $S_i$
```

2.5 Chapter Conclusion

In this Chapter we addressed the issue of sensor deployment. Special emphasis was placed on the unreliability of the network layout during the lifetime of the network, either due to malfunction or initial location placement of nodes during deployment. Reconfiguration and re-organisation algorithms have been proposed to address the above problem. However it has been seen that these solutions assume location aware nodes. To date, it has not been determined how much of an impact the location inaccuracies have on these algorithms, and it is in this regard that present study intends to contribute.

Chapter 3

Background Information – GIS and Simulation

"Geography is just physics slowed down,
with a couple of trees stuck in it."

T. Pratchett - The Last Continent

In the previous chapter the properties of WSAN deployment were outlined, included in these properties is that of location. The chapter introduced the possible dependency the reconfiguration and deployment algorithms have on location being accurate. An effective approach to studying the robustness of WSAN reconfiguration algorithms is to place nodes in the field and observe their coverage when their reconfiguration is based on location inaccuracies. This approach is costly and impractical. The next best method would be to place the sensors within a virtual world with the same obstacles and objects seen in the real-world.

The present chapter introduces Geographical Information Systems (GIS), key GIS principles, as well as a proposed GIS based simulation tool to assess the impact of location on various self-organising reconfiguration algorithms.

Section 3.1 discusses GIS and its position within this study, followed by a discussion on the usage of GENS[5] Smallworld for simulation. Smallworld is used as a development platform for the simulator as well as a repository to store data related to the GIS environment and to the simulator. Section 3.2 addresses the simulator and the architecture used to develop the tool. Section 3.3 details a simulation model and assessment of the algorithms. An explanation on the Smallworld Core program, how to use the application and the interface, is omitted from this work as it is deemed to be outside the scope of this study.

3.1 GIS Aided Simulation Platform

For the purpose of examining the extent to which the deployment and reconfiguration algorithms are capable of reducing coverage holes, this study proposes the development of a simulation tool based on a GIS application. Building on the discussion of the previous chapter, the simulator has to take into account distance and location. By integrating the environment with that of a GIS application,

[5] GENS – a division of General Electric that maintains and manages the GIS application created by a company called Smallworld, now acquired by GE. The name Smallworld continues as the product name and not the former company (GENS 2003).

real-world distance and scaling can be applied. A further benefit is that GIS systems come with a large library of mathematical classes, specifically in the field of geometry. These libraries assist in the overall implementation of the reconfiguration algorithms.

3.1.1 Geographical Information Systems

A Geographical Information System or GIS, is an information system capable of capturing, storing, analysing and managing data, data that is spatially referenced to the Earth[6]. The usage of GIS technology is widespread, including scientific research, environmental impact assessment, asset management, telecommunication network inventory management, urban planning and cartography to name a few (Huxhold 1991).

Figure 3-1 represents a brief history leading up to the development of current GIS systems. A GIS system by definition is a solution that associates object attributes to that of a graphical map or picture (Huxhold 1991). These attributes could contain information such as population density, commercial spending habits or even telecommunication infrastructure. If we apply this definition of a GIS system to history, then the first traces of such a system can be seen as far back as ±20000BC in Lascaux, France. Cro-Magnon hunters drew rock paintings of local animals with the migratory patterns linked to each artwork (Frazee 1997). Much later John Snow[7] mapped the 1854 cholera outbreak in London, England. Snow pinpointed all the reported cases of cholera, water sources and city streets on a map of London. The data assisted him in locating the source of the outbreak, contaminated water pumps, one of which later became known as *Snow's pump*, at Broad Street station (Vinten-Johansen 2003).

The time between 1969 and 1982 saw the birth of the modern day GIS system, in other words a system using computers to assist with spatial data analysis and storing. This sudden growth in GIS was spurred mainly by the progression of computer hardware and software development. During this time period, institutes such as ESRI (Environmental Systems Research Institute) and CGIS (Canada Geographic Information Systems) were founded. These institutes established

[6] Expansion of this definition to include other planets and natural satellites such as the moon is underway. An example of this is Google Mars^{TM}, an extension on Google Earth^{TM}.
[7] John Snow – 1813 – 1858, British physician and leader in medical hygiene (Vinten-Johansen 2003).

common standards and commercial acceptance of GIS tools (Huxhold 1991, Johnston 1990).

The last decade has seen an exponential growth in GIS based systems with the release of Earth Viewer in 2004, later acquired by Google and renamed as Google Earth in 2005, the founding of OpenGEO, a consortium aimed at standardising and approving open-source geo-information systems and application, in 2006. Finally various companies releasing web based tools allowing other applications to build on the GIS properties offered – such companies being eSpatial's iSmart, AutoDesk's Map 3D and MapGuide, Microsoft's Live Maps and GE MapFrame used on mobile devices.

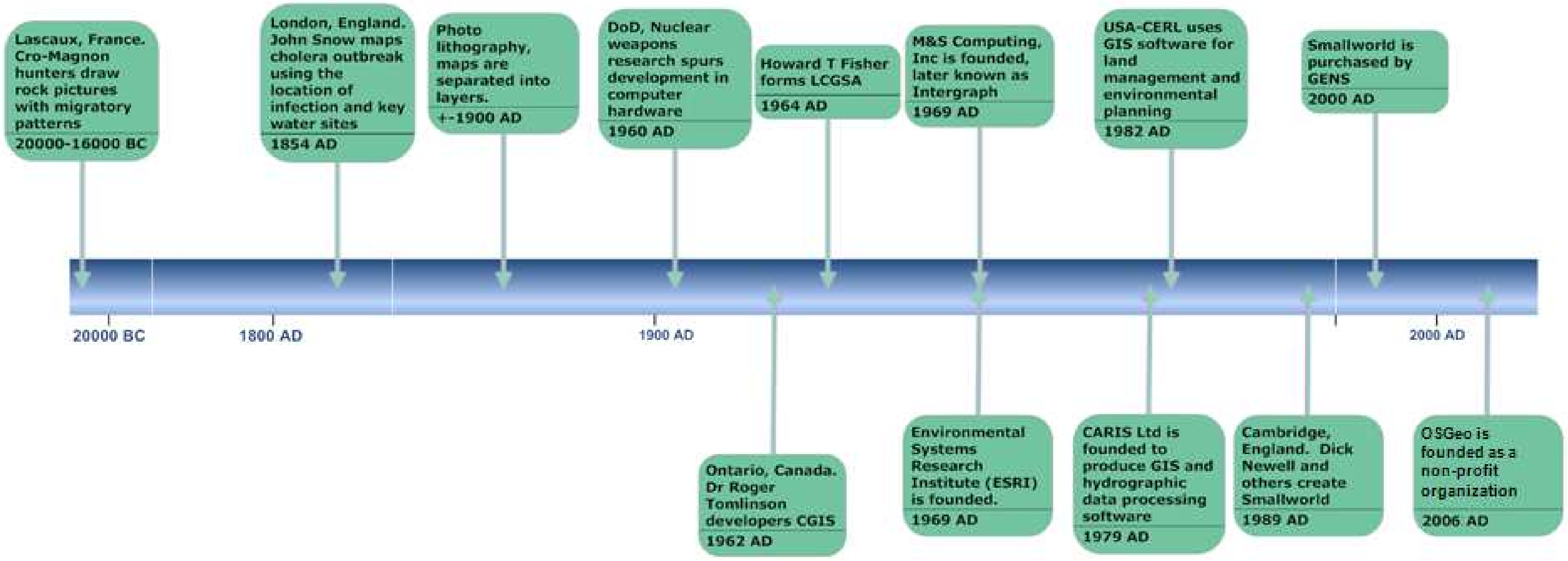

Lascaux, France. Cro-Magnon hunters draw rock pictures with migratory patterns
20000-16000 BC
London, England. John Snow maps cholera outbreak using the location of infection and key water sites
1854 AD
Photo lithography, maps are separated into layers.
+-1900 AD
DoD, Nuclear weapons research spurs development in computer hardware
1960 AD
Howard T Fisher forms LCGSA
1964 AD
M&S Computing, Inc is founded, later known as Intergraph
1969 AD
USA-CERL uses GIS software for land management and environmental planning
1982 AD
Smallworld is purchased by GENS
2000 AD
20000 BC
1800 AD
1900 AD
2000 AD
Ontario, Canada. Dr Roger Tomlinson developers CGIS
1962 AD
Environmental Systems Research Institute (ESRI) is founded.
1969 AD
CARIS Ltd is founded to produce GIS and hydrographic data processing software
1979 AD
Cambridge, England. Dick Newell and others create Smallworld
1989 AD
OSGeo is founded as a non-profit organization
2006 AD

Figure 3-1 - Timeline to the development of modern GIS solutions.

In 1989 Dick Newell founded the Smallworld company in Cambridge, England. The company's key role was developing a GIS application to be used by utilities companies, such as the gas, telecommunications and electrical industries. The Smallworld Company was later acquired by General Electric in 2000. The company only exists as the name of the GIS suite of products supplied by GE today. Finally in 2006 the Open Source Geospatial Foundation (OSGeo) is founded, a non-profit organisation providing financial, organisation and legal support for free and open source geospatial software. OSGeo leads the way to open source GIS software such as OpenStreetMap, GRASS GIS and OpenLayers (FOSS4G2008 2008)

A GIS application allows objects to be modelled in much the same way as any other database oriented application. In the case of the simulation tool, these objects could take the form of the network sensors, regions of interest, buildings, roads and even vegetation.

Common objects such as coast lines, buildings (land use) and vegetation can be grouped into layers. These layers can then be grouped further within a map topology and finally placed on top of a picture layer, Tagged Image File (TIF), or photo file.

Figure 3-2 shows various layers being placed over a map.

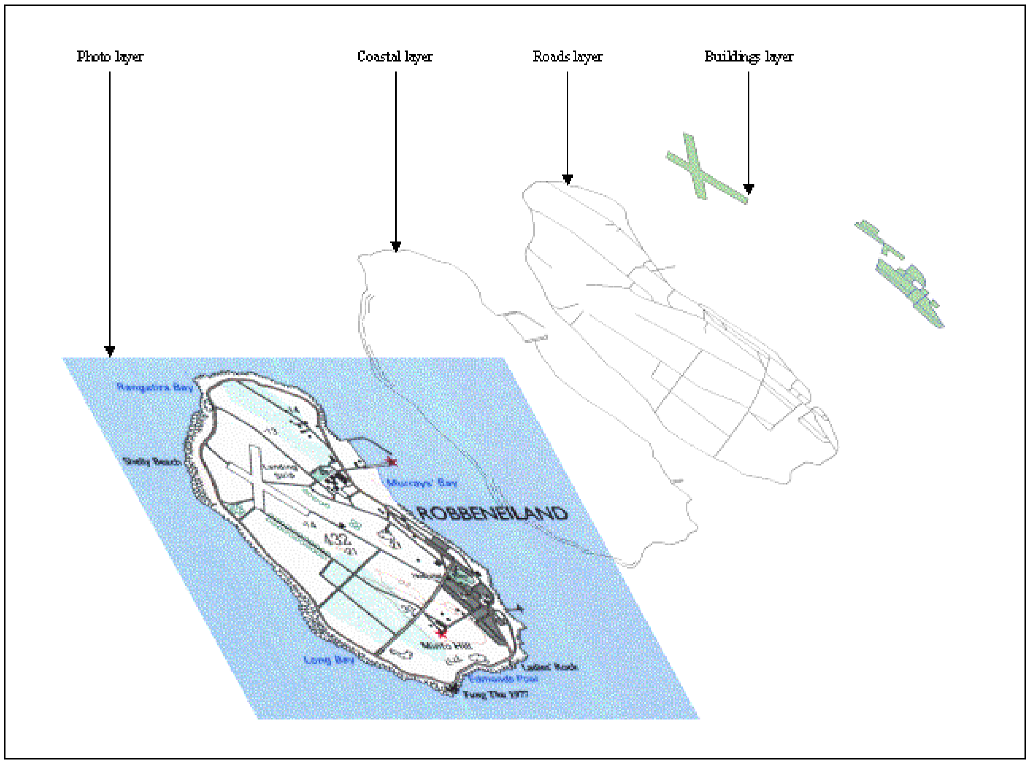


Figure 3-2 - Layers in a map topology.

A key difference with GIS applications is the way the data is represented. There exists a fair amount of specially designed databases to deal with geo-spatial data. Databases such as Oracle 10g spatial, Hibernate Spatial, GoeView, Siro-DBMS and VMDS are all equipped to store the data in such a manner as to make the data retrievable via *geo-spatial queries* (Gűting 1994, GENS 2003). Queries that are created with space in mind, for example the distance two objects are from one another, the area covered by an object or whether or not two objects interact spatially. Within the context of our sensor network deployment, the map topologies imported into the Core application are from the ESRI suite of standards. The map represents the Western Cape area of South Africa. The topology includes layers representing land use, man-made infrastructure - buildings and roads, and finally the natural landscape, for example the coastlines. Other objects such as the sensors are represented using Smallworld data types and are modelled directly within the Smallworld database.

3.1.2 Data Representation

In order for any GIS application to model the real-world environments, objects and data must first be modelled. Real-world objects can be represented in two distinct forms; discrete objects, such as parcels, houses, streets etc: and continuous fields, such as rainfall or land elevation.

The data is stored in the database as either a raster data type or vector data type. Collectively the data types are often referred to as an image (Huxhold 1991). With raster data, the layer is divided into rows and columns forming cells. Each cell contains data or information about that specific area, such as land use, population etc (Huxhold 1991) - Figure 3-3 (a). Vector data on the other hand consists of geometrical shapes: points, lines and closed polygons (Huxhold 1991). Each of these shapes is associated to data in a database, a row in relational databases or an object in object-oriented database. These shapes are specified with an instance of a co-ordinate – a location in other words - Figure 3-3 (b). Each co-ordinate is represented in terms of a co-ordinate system (Huxhold 1991).

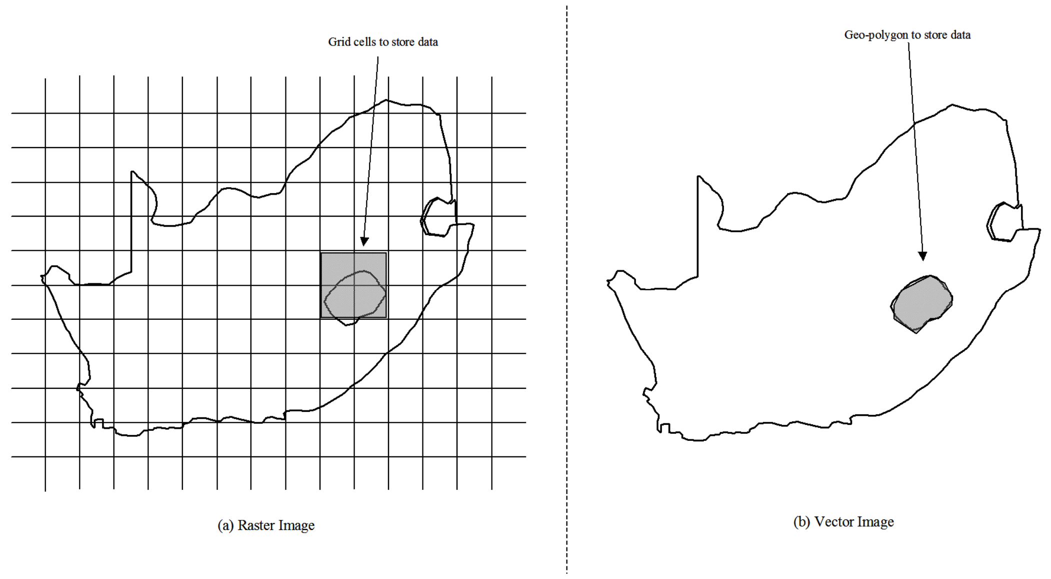


Figure 3-3 - Comparison of Raster and Vector image data-types.

A co-ordinate system can be defined by two different notations: a spherical system - Figure 3-4 (a) - using co-ordinates of latitude and longitude; or using a decimal point notation, covering a rectangular surface, where each rectangle is a small area of the Earth’s surface - Figure 3-4 (b).

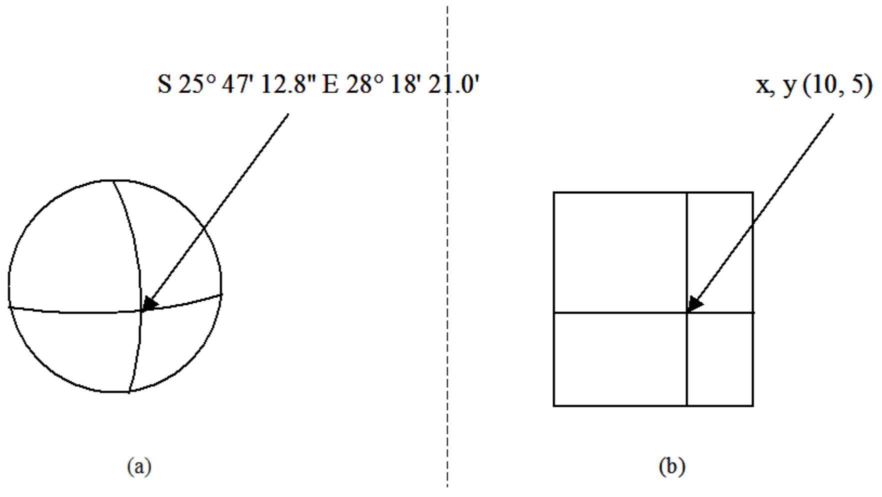


Figure 3-4 - Spherical Co-ordinate System vs. Rectangular Co-ordinate System.

For any GIS application to understand a valid co-ordinate system, a *geographical world* must be set up. A geographical world, as defined by GE Smallworld, is an object that contains a projection that includes the scale of the image (Huxhold 1991, GENS 2003). The projection scales the topology to fit within a given co-ordinate system (Huxhold 1991). Huxhold discusses three of the commonly used projection systems used within today's GIS applications: *Lamberts conic projection*, *Transverse Mercator projection* and *Stereographic projection*.

A projection can be understood by considering the result of unwrapping an ink-stained soccer ball covered in wrapping paper. The resulting image on the paper would then be the projection of the ink-stains on the ball. It would be simplest to wrap the ball in one huge sheet of paper, folding the paper in places to make it fit snugly around the ball. What if the wrapper wishes to wrap the ball with as minimum folds as possible? If cutting the paper were an option, the wrapper could stick small strips of paper, one at a time on the ball, thus minimising the folds. When the ball is unwrapped later, the paper can be stuck back together on a flat surface with no folds and creases. The folds in the paper represent errors within the projection; when the paper is folded, parts of the paper are not visible anymore. The same applies to maps being projected. The Earth is a geo-sphere much like the soccer ball. Thus when an aerial photo or map, known as a *geographical topology*, is created of a given area, the map is created with the contours and shape of the Earth. Small snap shots of the Earth are taken from the Earth's surface. These sections are stretched and skewed according to a given mathematical algorithm (known as the projection) and laid flat.

Without a valid projection, a view of the Earth would display Greenland double in visual size and a continent such as Africa would shrink. Thus the importance of an accurate projection system within the context of this study can be seen. The projection allows us to determine valid locations and distances, key aspects to this assessment. The locations in conjunction with the GIS mathematical libraries assist in the implementation of the reconfiguration algorithms. By using industry tested libraries, inaccuracies in the analysis are omitted.

In keeping with the Robben Island example for the ROI, a map of the Western Cape region of South Africa is projected onto our geographical world. The co-ordinate system used within this study is that of a rectangular surface area. The benefits of using a rectangular system are that the system is read as if it were placed within a Cartesian plane with *X* and *Y* float values representing a position of an object within the real-world. This system makes the co-ordinates easy to manage mathematically as well as being visually comprehensible for the user.

The projection applied to this area is that of the Lambert conical projection. Conical projection is the process of creating a cone like shape around the Earth. The cone is then unwrapped and laid flat to represent the area in a 2-dimensional state - Figure 3-5 - (GENS 2003).

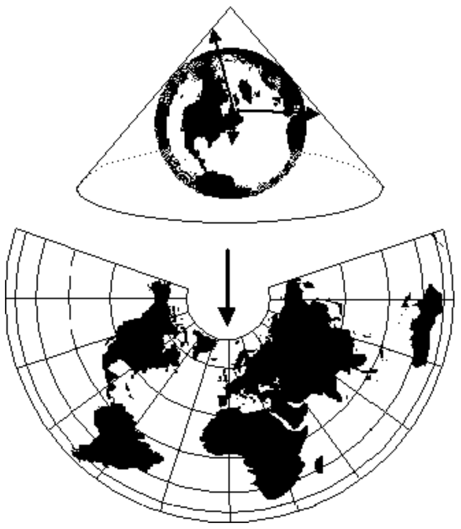


Figure 3-5 - Representation of conic projection (GENS)

The chosen GIS application on which to implement the simulator is that of GENS Smallworld Core. The reasons for choosing this application is the ease with which a system can be extended using the object oriented programming language, *Magik*.

3.1.3 GENS Smallworld Core Application

Smallworld is a GIS suite of products written for General Electric Network Solutions, GENS. The GIS platform is based on two GENS technologies.

The first, the object oriented programming language, Magik. The language is loosely based on that of Smalltalk, supporting multiple inheritance, polymorphism and a dynamically typed variable set (Yearsley *et al.* 1994, Wachowicz 1999).

The second is a proprietary database called Version Managed Data Store, VMDS (Wachowicz 1999, GENS 2003). The database has been developed with spatial technologies in mind. Optimisation is done for storing and analysing complex spatial queries. A spatial query is a query that selects data based on geometric principles. For example, does a geometric polygon intersect another polygon or is one polygon contained within another. These properties allow for a simple management of objects within a 2-dimensional world. The objects are then extended to include their spatial properties, location, direction and distance.

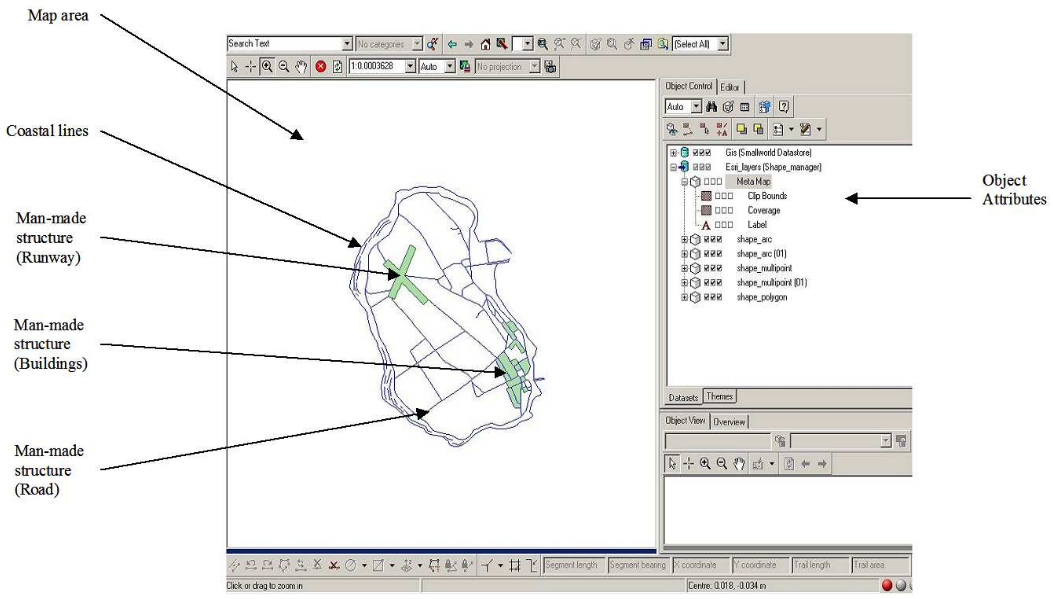


Figure 3-6 - Smallworld Application containing a topology of Robben Island.

The above figure shows the Smallworld Core application's GUI. Contained within the map area is a scaled map topology of Robben Island. The topology contains information regarding the land use, shown in dark, as well as the natural landscape such as the coastal lines.

Each object is represented as a polygon where each polygon has a type property, for example the objects runway, road and building are all polygons with a type runway, road and building associated to them respectively. Each polygon is also made aware of other polygons within the topology, this awareness is brought forward as a database relationship to other records in the table. Objects can be modelled with the same attributes as that of its real-world equivalent, thus modelling a virtual region for deployment.

By using a valid co-ordinate and projection system, Smallworld is able to place the objects to an accuracy of 1mm within this virtual world.

The following section introduces the development side to the simulator. In other words it discusses how the Smallworld Architectural Framework (SWAF) was used to develop the tool, as well as how the simulator may be extended to support further development.

3.2 Proposed Architecture

Appendix A.1 includes a table derived from Becker 2007 that compares all known WSAN simulators currently within the realm of research. Although some of the tools take into account the effects of radio interference – a further assumption of these algorithms - none of them deal with location.

Previously it was suggested that a GIS system such as Smallworld could be used to replicate the behaviour of real-world objects and their properties. We propose the development of a simulation tool that uses the properties and libraries of the GIS environment to accurately place nodes within a given target area.

The simulator, developed using object-oriented principles in Magik, is built as an application extending the GENS Smallworld Core application. Communication with the Core application is done via the *databus* - Figure 3-7. By communicating with the Core application, the simulator is able to retrieve all the GIS properties of the map and objects contained within the map. The databus uses a structure of *publish* and *subscribe*. This implies that the simulator sends requests via the databus to the Core application to perform tasks such as updating of the map, or retrieval of object locations, status and other properties. These requests are pushed onto the databus by the simulator; the Core application in turn has registered as a subscriber with the databus – when messages are sent across the database all subscribers registered with the databus listen for those specific messages. When the data is passed to the Core application, the data is picked up off the bus and handled accordingly.

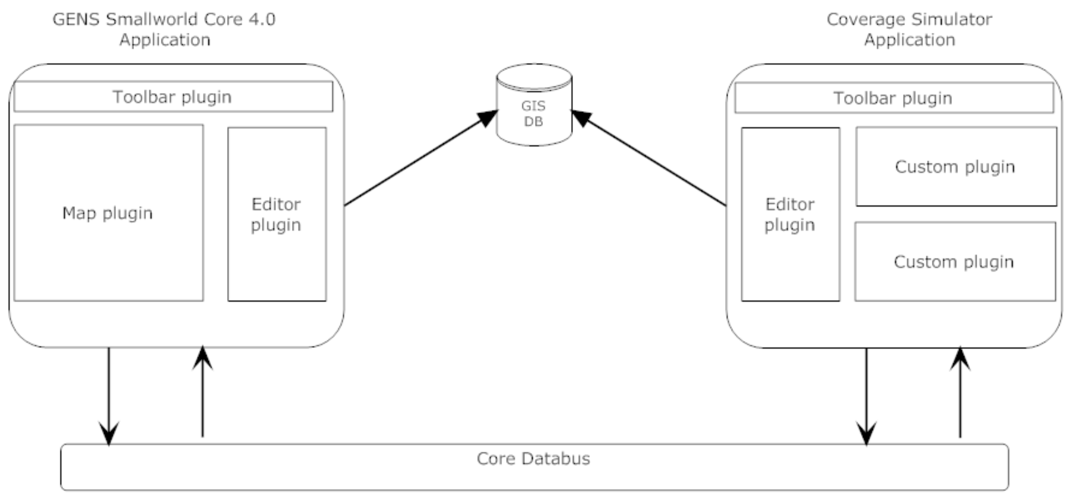


Figure 3-7 - Simulator application architecture

The modelling of objects within the Smallworld database is done via a Computer-aided Software Engineering (CASE) tool. The tool assists in creating objects, associating attributes, and associating *exemplar* class files to objects. The exemplar class files are Magik files that contain code to extend the functionality and logic of each object. In other words objects are implemented via the exemplar, much the same as a Java class file (Wachowicz 1999). Exemplars can be mapped directly to a table within VMDS, extending behaviour of the object that the table represents (Wachowicz 1999).

Figure 3-8 outlines the CASE tool data model for the set of objects used by the simulator.

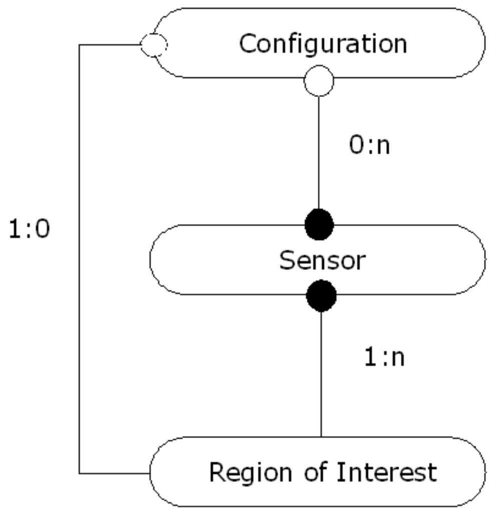


Figure 3-8 - Simulator data-model

Each sensor has a one-many relationship with that of a ROI. Thus, a single sensor can be contained by many ROIs. The ROI is modelled as a geometric polygon containing nodes and having a configuration. A ROI contains a single configuration of nodes. The configuration object includes the attributes for setting up the ROI and the placement of nodes within the network. Default values for sensing range and communication range are also specified by the configuration. Each of the database objects is mapped to a Magik exemplar.

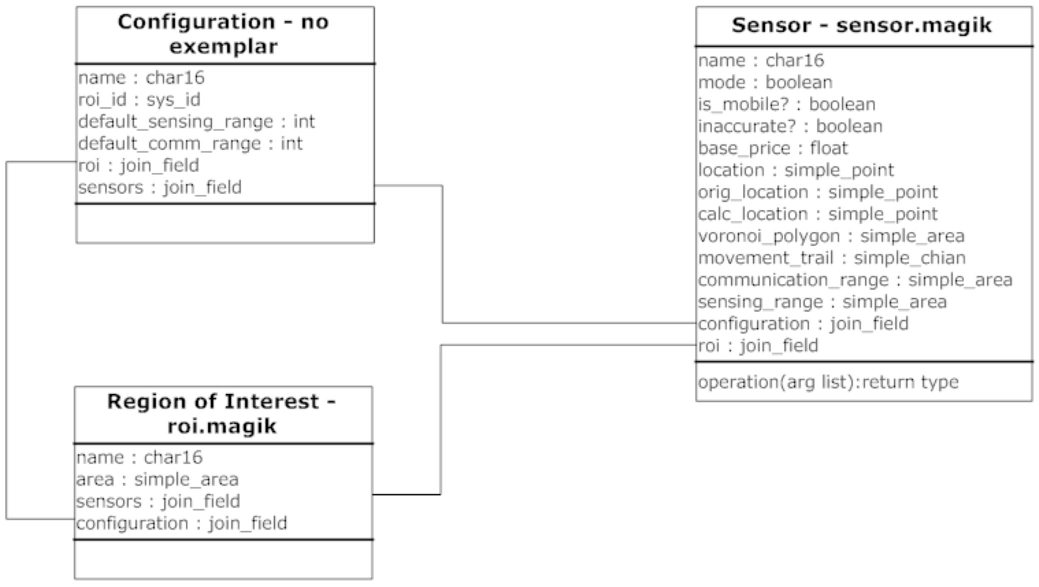

Configuration - no exemplar
name : char16
roi_id : sys_id
default_sensing_range : int
default_comm_range : int
roi : join_field
sensors : join_field
Sensor - sensor.magik
name : char16
mode : boolean
is_mobile? : boolean
inaccurate? : boolean
base_price : float
location : simple_point
orig_location : simple_point
calc_location : simple_point
voronoi_polygon : simple_area
movement_trail : simple_chian
communication_range : simple_area
sensing_range : simple_area
configuration : join_field
roi : join_field
operation(arg list):return type
Region of Interest - roi.magik
name : char16
area : simple_area
sensors : join_field
configuration : join_field

Figure 3-9 - Custom database objects with associated attributes.

Figure 3-9 shows a UML diagram for each database object. Shown in the diagram are the attributes and the data type of each object as well as the exemplar file associated to the object.

The sensor object has the properties of location, movement, sensing and communication distance as well as its local Voronoi polygon. Movement is stored as a Boolean data type, by storing movement as boolean, the property of movement can be turned on or off. Location is defined as a geometric point. The Voronoi polygon, sensing and communication attributes are represented as geometric polygons. Geometric polygons and points are objects containing geographical properties such as location, distance and area. The sensor object is extended by the sensor.magik exemplar, which provides additional logic to the database object.

The configuration object is responsible for storing different deployment scenarios within a given ROI. The configuration object contains a join field to the sensor object, as well as to the ROI object. A configuration object has default communication and sensing ranges, both are represented by an integer value. These values are used during the deployment of sensors where the communication and sensing range is uniform across all the sensors. Lastly a configuration has a unique name used for identification.

The ROI object is a simple geometric area. The object is aware of size and area as well as the sensors that have been deployed within its self. As with all other objects that are modelled, the ROI has a unique name for identification within the database. The geometric properties of the ROI object assist the simulator in calculating the presence of coverage holes by using spatial queries. The ROI object is extended by the roi.magik exemplar, providing methods to calculate coverage, coverage holes, node activity and algorithm results.

As GIS applications use images and shapes to represent data, it was deemed appropriate to develop a user interface to allow for the real time visualisation of the sensor objects moving within the ROI – Figure 3-10 shows the map of Robben Island as seen in Smallworld. The ROI area is enlarged to show how the sensors location changes over three iterations as a reconfiguration algorithm is run.

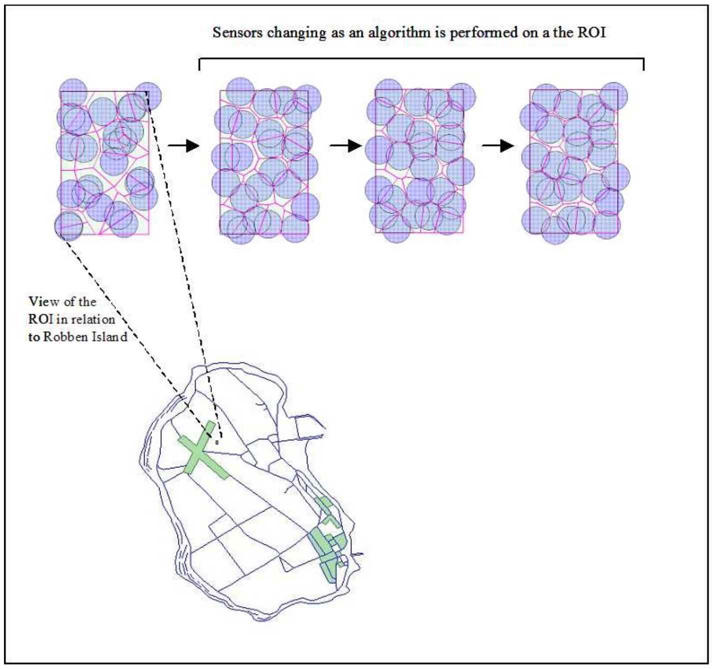

Sensors changing as an algorithm is performed on a the ROI
View of the ROI in relation to Robben Island

Figure 3-10 - An example ROI in relation to the map of Robben Island as seen during sensor reconfiguration

3.2.1 Simulator Interface

The simulator interface is developed using SWAF. SWAF has a software development architecture used by Smallworld for the development of applications. The framework uses the concepts of reusable *plug-ins* and *modules.*

SWAF consists of a *2-tier architecture*. The first tier, the presentation layer, contains the plug-ins, the second is a combination of the logical and persistent layers containing the database as well as the module *engines* (GENS 2003). The SWAF architecture defines an application as a standalone graphical interface that consists of its own set of functionality, toolbars and plug-ins (GENS 2003). A plug-in is defined as a lightweight object that provides top-level access to the application, for example an action button to trigger an event in the database (GENS 2003). Each plug-in makes a call to an engine for the processing of the GUI request. The engine contains all the business logic – this can be compared to Java Beans used by the J2EE development environment. Figure 3-11 shows the simulator, consisting of five plug-ins within the application.

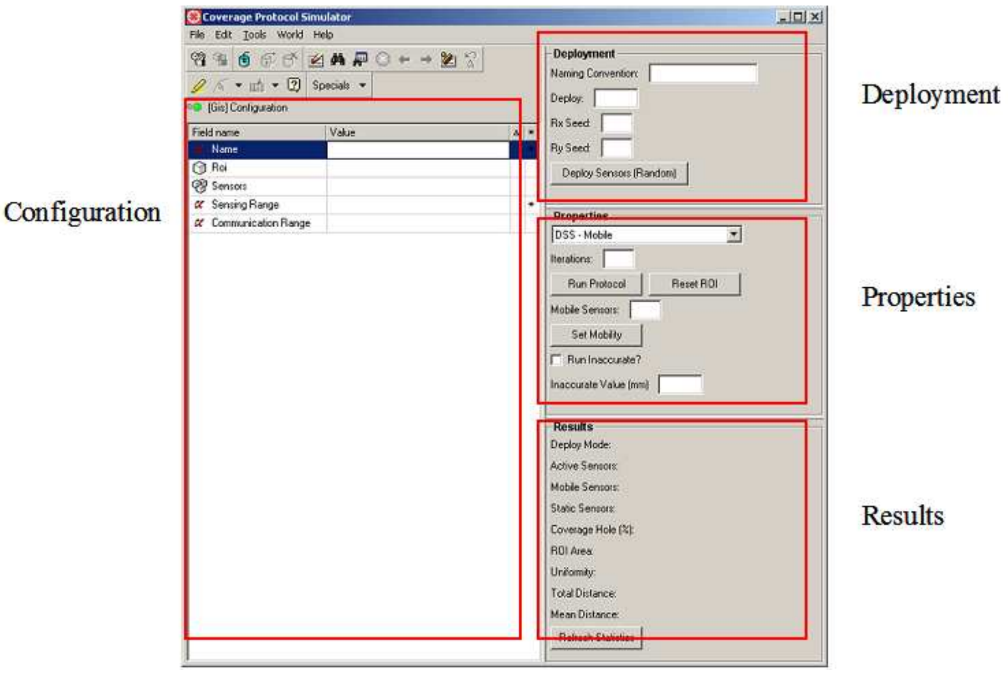


Figure 3-11 - Proposed Simulator Application.

The simulator GUI gives the user the option to create, modify or delete a configuration object. This is done via the *Configuration* editor. New nodes can be added to the ROI via the *Deployment* plug-in. The deployment plug-in allows the user to specify the naming convention used in the creation of new nodes as well as the degree of randomness that should be implemented during the deployment process. By changing the deployment seed values of either the *X* or *Y* axis the user has the ability to deploy nodes sparsely or densely clustered within the ROI. A second benefit of specifying the seed values is that the deployment setup (randomness) of the nodes can be replicated. All the newly deployed nodes are deployed within the selected configuration. The default deployment of nodes is in the mobile state.

Running of algorithms is done via the *Properties* plug-in. A drop down list is provided for the selection of the algorithm to be run on the specified configuration. The user is given the option to run the algorithm a given number of times or iterations. Thus acting as a terminating criterion for the algorithm. Terminating conditions are discussed in further detail in Chapter 4.

Depending on the type of algorithm to be run, an option is given to set a given sub-set of the nodes to be mobile or static.

Finally the user has the ability to invoke inaccuracy upon the node locations. All the nodes contain an attribute for '*inaccurate location*'. By setting the inaccurate checkbox the inaccurate location value is used throughout the process.

Results of the algorithm are displayed to the user in the *Result* plug-in. Within this plug-in the user is able to see the properties setup for the algorithm as well as the total coverage hole percentage, total movement and mean movement of the sensor nodes. The application has the option to export the data to Microsoft Excel. The exported data contains the configuration setup, node names and data associated with the configuration for all iterations including the coverage hole, distance travelled and mobility status for all nodes.

3.2.2 Development of Algorithms

All of the coverage algorithms are implemented within a single module within Smallworld. A module in terms of Smallworld can be compared to a package in Java. The coverage_protocol exemplar is an abstract class that contains the methods required by the simulator to run, reset, stop and pause the algorithm. These methods are used for all coverage algorithms. Further abstract methods are added by children classes depending on the type of algorithm developed - Figure 3-12. Searching for classes inheriting from coverage_protocol allows the simulator to detect a new algorithm.

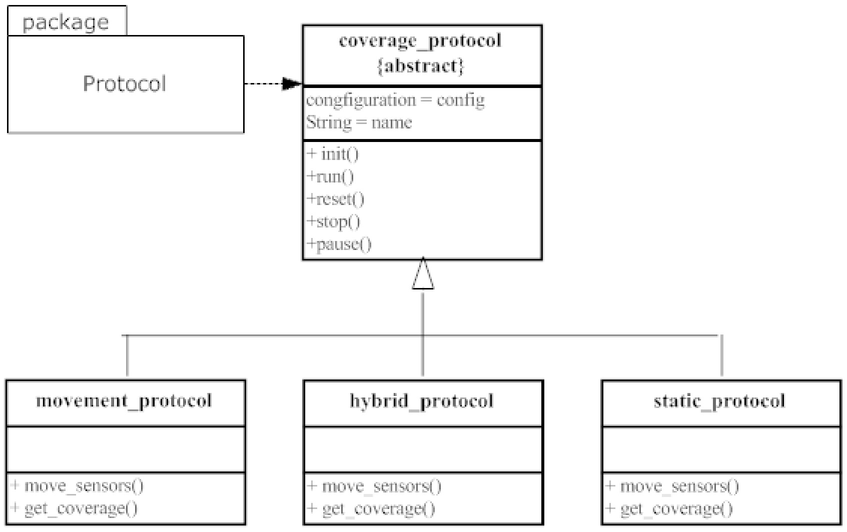


Figure 3-12 - Class diagram for the development of new coverage algorithm.

An outlined API for each component to the simulator is covered in Addendum A - Development API.

Further modules used in the development of the simulator include: voronoi_diagram, a module used in the computation of Voronoi polygons and coverage holes. This module contains exemplars and methods for calculating the Voronoi polygons and diagrams for a given set of nodes as well as the existence of coverage holes; sw_core, a module used for enhancements to the Smallworld Core application, such as modifications to the database and map classes; finally the geometry module, a module for common mathematical methods used by the protocols or simulator engine exemplars.

Figure 3-13 shows all the exemplar files developed for the simulator. This excludes the exemplars supplied by the Smallworld Core library. For a detailed description on the methods refer to the Development API Addendum.

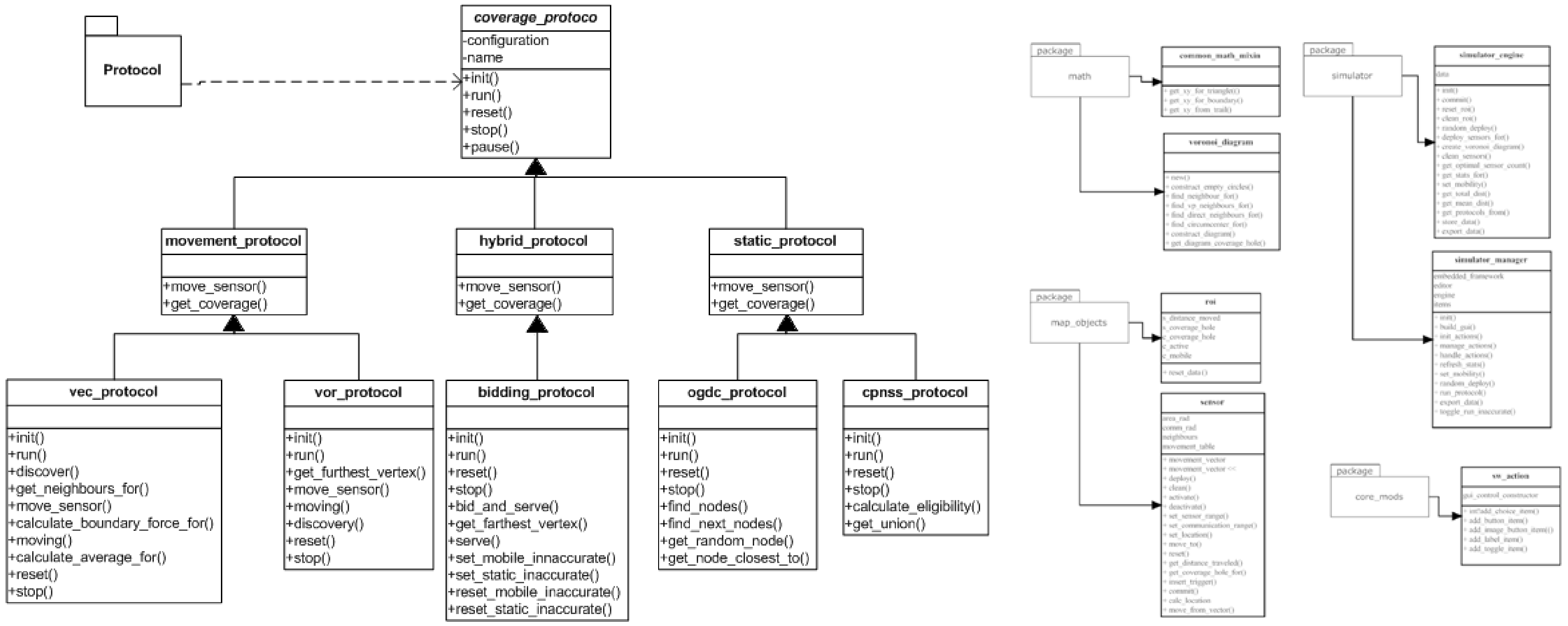


Figure 3-13 - UML diagram containing all exemplars within the simulation application.

3.3 Proposed Assessment

It has been established that location plays an important role in the reconfiguration or self-organising WSAN algorithms. In the introduction we brought forth two research questions. Both of the questions are centred on the assessment of location accuracy in the deployment and reconfiguration of the networks. We have also seen the benefits of using the properties of GIS to aid in replicating WSAN deployment. We propose to use these properties of the GIS environment and the simulator to assess the sensitivity of the various coverage algorithms to the quality of the node location data. The simulator allows for repetitive experiments to assess each algorithm.

3.3.1 Experiments

The simulation exercises first address the algorithms within a perfect world. By this is meant that no inaccuracies, interference or malfunction of nodes is introduced. Further experimentation is performed where inaccuracy is introduced into the world. The network is subjected to gradual increase in the number of inaccurate nodes, followed by a worse-case scenario, *high-inaccuracy*, of all the nodes being inaccurate. Each of the algorithms contains a termination criterion. The termination criterion prevents the algorithms from running indefinitely. This criterion can take the form of a given number of runs or even an acceptable degree of coverage. The current simulation models take into account two degrees of termination. In the first case, the termination criterion is set at an acceptable real-world level. In the second case, simulations are performed with an unrealistic termination criterion, forcing a much larger number of runs then needed within the real-world.

Various algorithm authors have assessed algorithms in terms of different properties. For example Jiang and Dou (2004) assess the static algorithms based on energy saving, whilst Wang *et al.* (2003a, 2004) assess their algorithms based on the degree of movement by each node and final coverage. For the sake of commonality, the algorithms tested here were assessed on the percentage of total coverage within the ROI after the algorithms have been executed. The following chapter, titled Methods and Procedures, addresses the experiments in greater detail.

Radio interference and node malfunction are two factors that may influence the network layout. However these are deemed to be areas of future research.

3.4 Chapter Conclusion

In this chapter we discussed briefly the history of GIS and how it may benefit WSAN simulation. We outlined a simulator tool to be built using GENS Smallworld Core as the underlying GIS application. The benefits expected of such a simulator are to aid in the study of how inaccuracies affect the overall outcome of the self-organising coverage protocols introduced in Chapter 2. Further studies can also be implemented using the simulator. Such studies are discussed in the concluding chapters.

The next chapter introduces and explains each of the five algorithms, followed by a discussion on the assessment experiments, used to re-evaluate each algorithm based on location inaccuracy and final coverage.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 4

Methods and Procedures

"Space is big. You just won't believe how vastly,
hugely, mind- bogglingly big it is.
I mean, you may think it's a long way
down the road to the chemist's,
but that's just peanuts to space."

D. Adams – The Hitchhikers
Guide to the Galaxy

We have seen the benefits of a GIS integrated simulation tool to model WSANs as if they were placed in the real-world. The present chapter discusses how the study assesses five deployment and reconfiguration algorithms used in WSANs. Section 4.1 discusses each of the experiments that are performed on the algorithms. The experiments take into account accurate and inaccurate location within the algorithm as well as terminating criterion. The next section, 4.2, discusses, in detail, each of the algorithms. The section includes basic pseudo-code for the running of the algorithms.

As the deployment of large networks is costly, the GIS environment allows for the set up of a virtual world containing real-world objects and network behaviour. By applying the network within this context the experiments can be performed on the algorithms as if they were being done in the real-world at a much less cost and time.

4.1 Experiments

For the purpose of assessing the five algorithms, five experiments were set up using the simulator. The first experiment, *control*, is used to assess algorithm performance in an ideal world with no inaccuracies. The second experiment confirms the results of the control experiments by testing the algorithms with different starting scenarios. The third, fourth and fifth experiments introduce location inaccuracy to the algorithm.

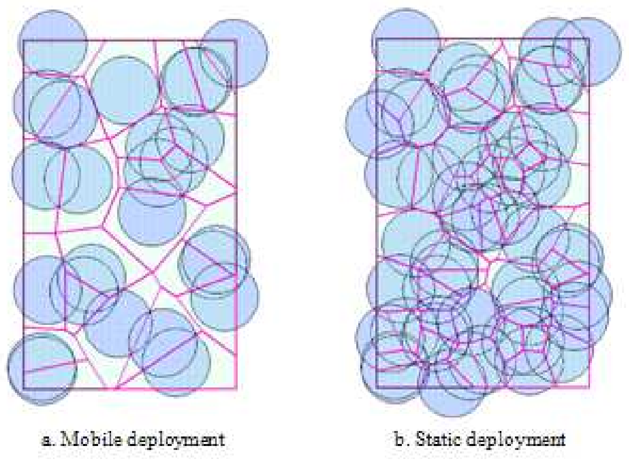


Figure 4-1 - Initial starting scenarios.

An environment is simulated in which 25 - 50 nodes are deployed randomly within an ROI that has an area 1615m^2. Each node was assigned a sensing range of 5 meters and twice that in communication range – i.e. 10 meters. The value of the communication range is derived from the study by Zhang and Hou (2003) where they prove that by setting the radio communication range to at least double the sensing range, a complete coverage of a convex area implies connectivity among the set of nodes within the ROI.

Depending on the deployment state of the network being static, hybrid or mobile, a percentage of the nodes were deemed to be capable of self-movement, and all to be unaware of power consumption. It was also assumed that all the nodes were deployed without failure, and that sensing and communication took place within the respective radii without interference.

All the algorithms run iteratively until a given terminating condition is satisfied. The terminating condition for the algorithms may be defined in various ways. For example: a given degree of coverage is reached; or no movement by nodes for more than one iteration. For the purpose of consistency, the terminating condition and experiment configuration is kept the same for all the algorithms. By configuration we mean the ROI, node count, layout of the network, sensing distance and communication range.

The environment setup depicts that of a potential real-world setup and represents the environments consistent with that described by the original authors of the algorithms.

4.1.1 Control Experiment

The control experiment is used as a benchmark environment, in other words an environment that contains no flaws, or errors, which is used as a basis for comparison. The results obtained from this experiment are compared to the results obtained by the algorithms' authors. All nodes are deployed into the given ROI and the algorithm run. During the control experiment, a real-world terminating condition is set. By real-world conditions, we mean termination such as limited energy or limited mobility (in the case of movement-assisted nodes).

4.1.2 Initial Deployment Sensitivity

The initial deployment of nodes within a ROI could potentially affect the final outcome of the algorithms' performance. For example, in the case of movement-assisted algorithms, when nodes are extremely clustered in one area, the algorithm needs more iterations to disperse the nodes. The following experiment aims at assessing the outcome of different random starting scenarios, in each case the random deployment limiting the nodes to the ROI. In contrast to the control experiment a high, unrealistic, terminating condition is set. Thus we are able to assess the algorithm performance long after real-world conditions would have caused termination. This allows for extended assessment to verify that the normal termination condition does not lead to premature termination.

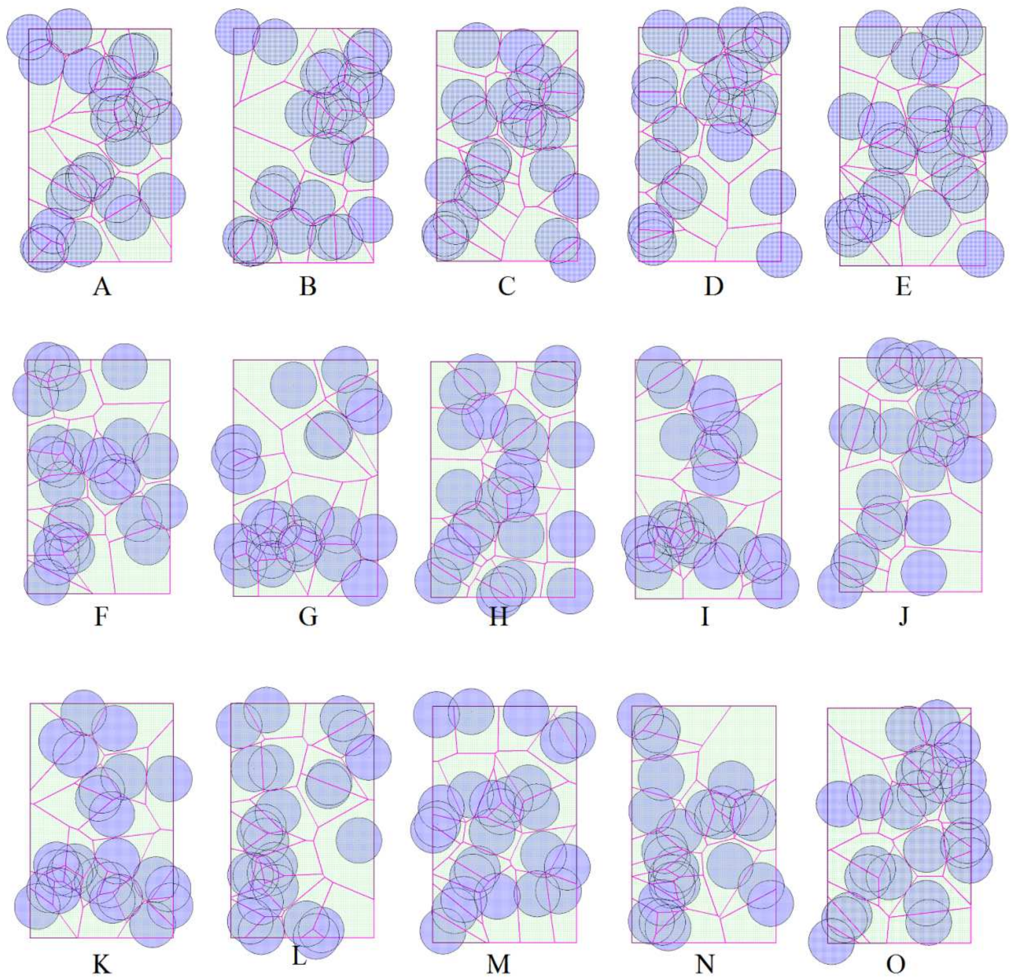


Figure 4-2 - Initial deployment of nodes under fifteen different scenarios for mobile and hybrid networks.

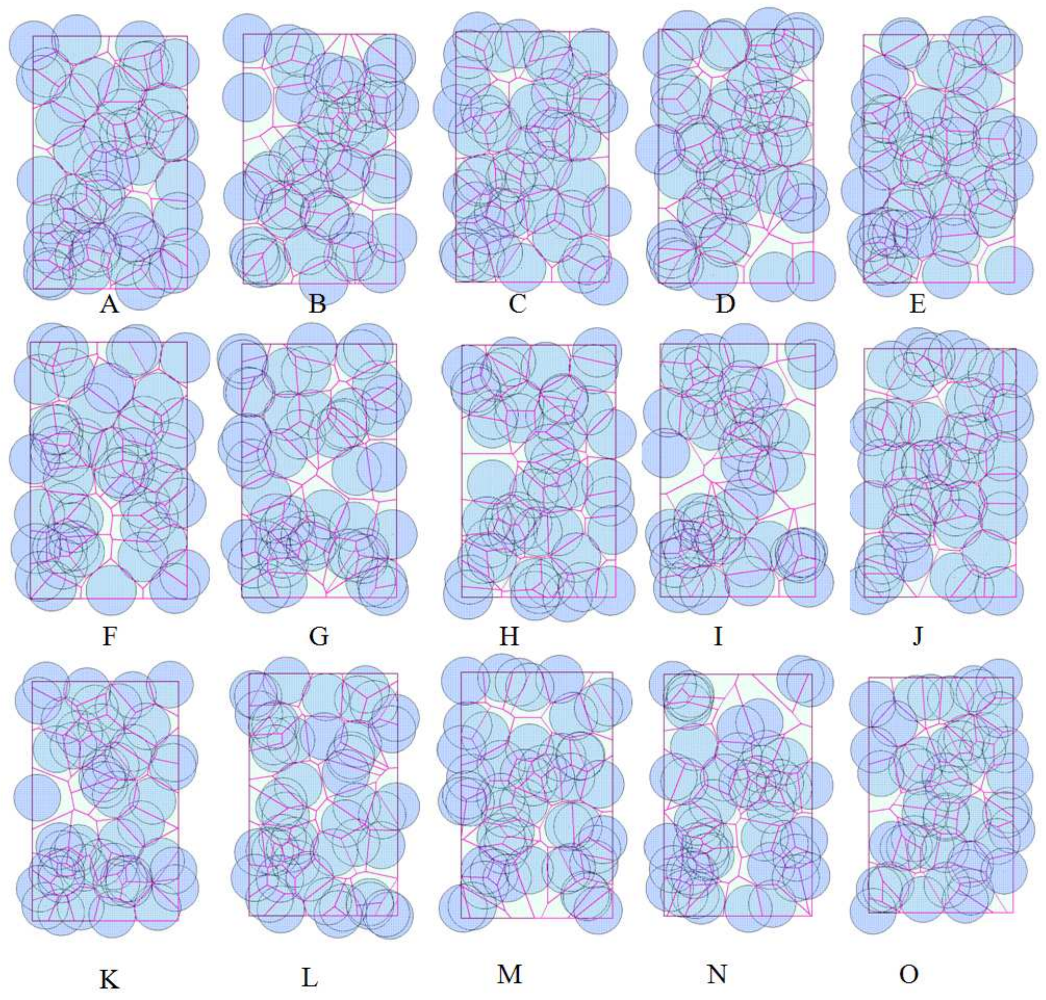


Figure 4-3 - Initial deployment of nodes under fifteen different scenarios for static networks.

Each algorithm's behaviour is tested from each of fifteen randomly determined initial deployment locations. The starting locations are kept constant for mobile and hybrid networks - Figure 4-2. However with static networks, a further 25 nodes are deployed into the ROI to allow for some redundancy within the network - Figure 4-3. The minimum, maximum and average coverage hole size is recorded for each iteration.

Using the statistical observation method of *outliers* the study will compare the results of each of the starting scenarios to one another at different intervals. The method used for identifying the outliers chosen is that of the Interquartile range (IQR). An outlier is defined as an observation that appears to be inconsistent with the other observations in the data set (Barnett and Lewis 1994). By observing these starting scenarios the study will be able to determine if there are any scenarios that deviate from the rest.

IQR is a measure of variability or statistical dispersion. It is the difference between the top 75%, (Q_3), and the lower 25% (Q_1). If the observed value aObv is within L_{lmt} and U_{lmt} then it can be said that the value observed for that starting scenario is within a reasonable limit of being similar to the other starting scenarios - Equation 4-1. If the observed value is outside L_{lmt} and U_{lmt}, then it can be said the value differs from the rest. *K* is the outlier coefficient, used to determine the degree of certainty in an unskewed distribution. In this case the value is left as the minimum of 1.5 (Barnett and Lewis 1994).

The final coverage hole value for each of the 15 starting scenarios is used as the sample rate to assess. By performing the assessment on the final states of the network, the experiment attempts to show that the final results do not differ significantly.

Equation 4-1

$$Q_1 = Percentile(aObv_n, 0.25)$$

$$Q_3 = Percentile(aObv_n, 0.75)$$

$$IQR = Q_3 - Q_1$$

$$Outlier\ lower\ limit\ L_{lmt} = [Q_1 - k.IQR]$$

$$Outlier\ upper\ limit\ U_{lmt} = [Q_3 + k.IQR]$$

4.1.3 Graduated Inaccuracies - Sensitivity to Inaccuracy I

Real-world inaccuracy is unlikely to occur in all nodes within the network. Thus, this experiment, *Sensitivity to Inaccuracy I*, simulates nodes whose actual positions are inaccurately reported to the respective 5 placement algorithms. In each case, the percentage of inaccurate nodes is progressively reduced within the network, thus assessing how the algorithms behave when the percentage or ratio of inaccurate nodes to accurate nodes decrease. This experiment assists in evaluating algorithm dependency on various degrees of the node inaccuracy.

The experiment is run on the same criteria as that of the next *Sensitivity to Inaccuracy II*. In this case however the percentage of inaccurate nodes is decreased gradually.

4.1.4 High-Inaccuracy – Sensitivity to Inaccuracy II

The following experiment, *Sensitivity to Inaccuracy II*, is used to test the algorithms effectiveness against worst-case inaccuracy, i.e. all the nodes in the network are inaccurate to the same degree. The same initial deployment setup as with the control experiment is used to run each algorithm with various levels of inaccuracy. Twenty simulations are run per algorithm. In each simulation, *all the nodes* are set at an inaccuracy level of *n*, where the inaccuracy ranged from 1 meter to 10 meters, in intervals of 0.5 meters. A realistic termination criterion is used to reduce the number of iterations. The *Initial Deployment Sensitivity* experiments are used to determine what the termination factor should be. Each algorithm is run 50 times per inaccuracy level, recording the minimum, maximum and average coverage holes. Each run allows for a different inaccurate location point for the sensor.

4.1.5 Termination Criterion Sensitivity

A final experiment is performed on each algorithm. The same criterion is used as that of high-inaccuracy and graduated inaccuracies experiments. However during this experiment, an iterative termination count of 100 is put in place. By this is meant that the algorithm terminates after 100 iterations. By performing this experiment we are able to assess the impact of location inaccuracies over an extended period of time.

4.1.6 Overview of Experiments

The following table provides an overview of the above mentioned experiments. For completeness the values related to the experiments such as initial number of nodes deployed, number of times the experiment is run, the number if iterations within each experiment and termination criterion are included – these are to be discussed in further detail in the following Chapter.

Table 4-1 - Experiment Summary

Algorithm	*Network Layout*	*Initial number of nodes deployed*	*Number of experiments run*	*Iterations per experiment run*	*Accurate : Inaccurate nodes*	*Termination Criterion*	*Placement Error*
Control	Mobile	25	1	4 (as a result of the termination criterion)	25 : 0	Mean distance 10m	None
	Hybrid	48 (25 static : 23 mobile)	1	4	48 : 0	When no mobile movement is made	None
	Static	50	1	1 (CPNSS)/ 50 (OGDC)	50: 0	None †	None
Initial Deployment Sensitivity	Mobile	25	15	100	25 : 0	None †	None
	Hybrid	48 (25 static : 23 mobile)	15	Varied – based on the algorithm, average ±5	48 : 0	When no mobile movement is made	None
	Static	50	15	1 (CPNSS)/ 50 (OGDC)	50 : 0	None †	None

Algorithm	*Network Layout*	*Initial number of nodes deployed*	*Number of experiments run*	*Iterations per experiment run*	*Accurate : Inaccurate nodes*	*Termination Criterion*	*Placement Error*
Inaccuracy Sensitivity I	Mobile	25	25	4 (as a result of the termination criterion)	Varied *	Mean distance 10m	Varied **
	Hybrid	48 (25 static : 23 mobile)	20	Varied – averaged ± 6	Varied *	When no mobile movement is made	Varied **
	Static	50	50	1 (CPNSS)/ 50 (OGDC)	Varied *	None †	Varied **
Inaccuracy Sensitivity II	Mobile	25	20	50	0 : 25	50 Iterations	Varied **
	Hybrid	48 (25 static : 23 mobile)	20	Varied – averaged ± 6	Varied *	When no mobile movement is made	Varied **
	Static	50	20	50	Varied *	50 Iterations	Varied **
Termination Criterion Sensitivity	Mobile	25	1	100	25 : 0	100 Iterations	None
	Hybrid	48 (25 static : 23 mobile)	n/a	n/a	48 : 0	When no mobile movement is made	None
	Static	50	n/a	n/a	50 : 0	n/a	None

† Due to the nature of the network topology the experiment shall terminate or converge at a given point without the need for a termination criterion.
* The ratio of accurate to inaccurate nodes is adjusted, after which the experiment is run again.
** The placement error is changed according to the level of inaccuracy needed to assess the experiment.

4.2 Algorithms

Two prominent algorithms from each of the three deployment scenarios were chosen for assessment, namely mobile, hybrid and static. The decisions to use the algorithms are based on the results presented by the authors of the respective algorithms. The presented results suggested that the chosen algorithms were the most efficient in terms of solving the coverage hole problem.

4.2.1 Mobile Algorithms

Here the so-called VEC and VOR algorithms are described as representative of a class of mobile node placement algorithms, i.e. a class that assumes that the nodes can be moved. Both algorithms are due to Wang *et al.* 2004. In the case of mobile algorithms, 25 nodes are deployed to the ROI – Figure 4-1 (a). By deploying a number of nodes that would result in large coverage holes within the ROI, movement is guaranteed.

4.2.1.1 VECtor-based Algorithm (VEC)

The VEC algorithm is a *push-based* algorithm in that it pushes the neighbouring nodes away from each other. The algorithm is inspired by the behaviour of electromagnetic particles: two particles exert a force on each, inversely related to the distance between them that pushes them apart (Wang *et al.* 2004).

Nodes are assumed to be optimally placed when they are evenly distributed within the ROI, each one being at some constant distance, d_{avg}, from its neighbours. Since the number of nodes and ROI size is known, this value may be pre-computed.

Suppose that $d(S_i,S_j)$ is the distance between sensor S_i and S_j. If $d(S_i,S_j) > d_{avg}$ and if S_j is within communication distance of S_i, then VEC assumes that S_i and S_j mutually exert a “virtual force” on one another that is proportional to $(d_{avg} – d(S_i, S_j)) / 2$. In general, this virtual force, cumulatively determined for node S_i, determines the distance and direction that S_i moves in each iteration of the algorithm.

However, there are a number of special considerations. In the first place, if S_i already covers its local area as defined by the Voronoi polygon, then it will not be moved. Instead, the force $(d_{avg} – d(S_i, S_j))$ will be exerted on the node S_j only. Secondly, to prevent the nodes from moving too close to the boundary, an additional force is generated by the boundary of the ROI. The boundary force will push the node to move ($d_{avg}/2 – db(S_i)$), where $db(S_i)$ is the distance of S_i to the ROI boundary. Refer to Figure 4-4.

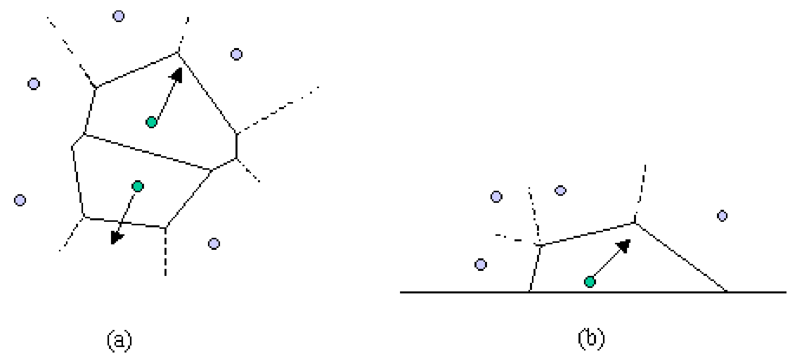


Figure 4-4 - (a) Virtual Forces between two sensors, (b) Virtual Force exerted by a boundary.

To prevent greater coverage holes from forming due to the movement of a node, Wang *et al.* (2004) introduce *movement-adjustment*. After the final virtual force on a node has been determined, the local coverage is recalculated based on the potential movement. If the coverage is not improved, a midway point between the node's current location and calculated location is examined. If the local coverage is increased at this new target location then the node is moved accordingly; otherwise the node remains in the current position for one iteration.

A further check is put in place to prevent the node from moving outside the ROI. If the node location is placed outside the ROI, then the node remains in the current position for one iteration.

The VEC algorithm thus runs iteratively until a given threshold is reached. Each iteration consists of two phases:

- a discovery phase, done locally by each node, in which the broadcasting of locations to neighbouring nodes as well as the calculation of the local polygons for each node takes place; and
- a movement phase, during which the movement of each node is determined. During this phase a new location for the node is determined. This location is deemed as a new optimal position for the node to be placed. By optimal it is meant a new position that best covers the Voronoi Polygon

Table 4-2 - VEC algorithm pseudo-code.

```
Notation:
  S = Sensor
```

N = set of neighbours

C_i = complete coverage of Voronoi polygon

V_i = moving vector for sensor

d = distance

d_{avg} = Average distance between sensors when the sensors are evenly distributed within the network

$\overrightarrow{V_f}$ = Force exerted, i.e. the distance to move away from the current position

Algorithm:

```
1. Enter discovery phase
      a. Broadcast current location to all nodes within the reachable network
      b. Calculate Voronoi-diagram
2. Enter movement phase
      a. Calculate the presence of a coverage hole
      b. If $C_i$ = false then
            i. $V_i = 0$, i.e. set the movement of the node to 0
           ii. Loop over all the neighbour nodes ($N_i$) of the current node ($S_i$)
                  1. Check if both the neighbour $S_j$ and $S_i$ should move
                        a. If true then $\overrightarrow{V_f} = ((d_{avg} - d(S_j, S_i)) / 2)$
                        b. Else $\overrightarrow{V_f} = (d_{avg} - d(S_j, S_i))$
                  2. Check if the boundary exerts a force on $S_i$
                        a. If true apply $\overrightarrow{V_f} = (d_{avg} - d(S_j, S_i))$ on $S_i$
                  3. Sum forces with $V_i$
          iii. Perform movement according to $V_i$
      c. else
            i. Skip current node
```

4.2.1.2 VORonoi-based Algorithm (VOR)

The VOR algorithm is a *pulled-based* algorithm in that it pulls the sensors to their local maximum coverage hole. The maximum coverage hole can be described as the largest area within a node's local area, as specified by the Vornonoi-polygon, which is not covered by the sensor. Once the node detects the existence of a coverage hole, the node then moves towards the farthest vertex of the relevant Voronoi polygon. The distance that the node moves, denoted as V_{far}, is calculated as the distance to the farthest vertex, less the sensing radius of the node, i.e. $V_{far} = d(S_i, A) - R_i$. In Figure 4-5, it is assumed that V_{far} corresponds to the distance from S_i to B, so that the node S_i should be moved to point B. However the distance moved is limited to be at most half the communication distance (as opposed to the sensing distance). This avoids situations where the node's local view of the Voronoi polygon does not account for the existence of a neighbour. Wang *et al.* (2004) argue that without such a limitation, the node might be moved too close to the neighbour's sensing area, thus distorting the result.

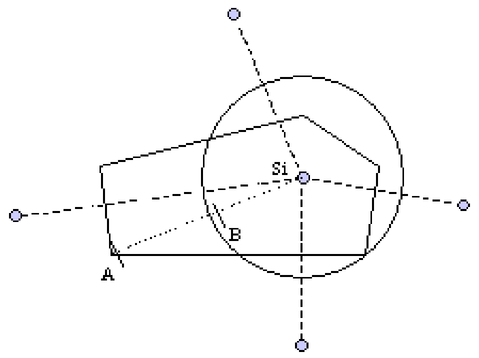


Figure 4-5 - Movement of nodes using VOR (Wang 2004).

VOR can be classified as a greedy algorithm in that it attempts to reduce the largest holes. However, the movement of a node to solve a hole in one direction could potentially cause another hole in the opposite direction, resulting in *movement oscillations*, so that the sensor moves back and forth continuously between two points. To prevent this, the VOR algorithm introduces *oscillation control*. The previous direction of movement of each node is stored. Each time that a node needs to move, a check is first made to verify that the next move is not in the opposite direction to the last move made. If this holds true, then the node remains in the current position for one iteration.

Due to the greedy nature of the algorithm, a second problem may occur: much the same as in VEC, the reduction of the size of one coverage hole may result in an even greater coverage hole appearing elsewhere. For this reason, a second movement adjustment rule is put in place. Before committing a node to a certain movement, the potential Voronoi polygon that would result if the move were to be made is calculated. If the resulting coverage hole is equal or greater than the current coverage hole, then the node remains in the current position for one iteration.

The VOR algorithm shares the same attributes as VEC in the sense that it runs iteratively until a given threshold is reached. Each iteration consists of the same two phases as VEC, a discovery phase and a movement phase.

Table 4-3 - VEC algorithm pseudo-code.

```
Notation:
  $C_i$ = complete coverage of Voronoi polygon
  $\vec{V_i}$ = moving vector for sensor
  $\overrightarrow{V_{i,f}}$ = vector from $S_i$ to $V_{far}$
  $d_{max}$ = maximum moving distance i.e. the communication range

Algorithm:
  1. Enter discovery phase
       a. Broadcast current location to all nodes within the reachable network
       b. Calculate Voronoi-diagram
  2. Enter movement phase
       a. Calculate the presence of a coverage hole
       b. If $C_i$ = false then
            i. Calculate $V_{far}$ as the furthest vertex of $S_i$
           ii. Calculate $\vec{V_i}$ as $\overrightarrow{V_{i,f}}$ -sensing range of $S_i$
          iii. If $|\vec{V_i}| > d_{max}$ then reduce $Vi = d_{max}$
           iv. perform oscillation control
                 a. Check if movement direction is opposite to that of the
                    previous round
            v. perform movement-adjustment
                 a. Check if local coverage will be increased if the movement
                    occurs
       c. else
            i. Skip current node
       d. Move sensors according to $V_i$
```

4.2.2 Hybrid Algorithms

The hybrid algorithm chosen for this study is that presented by Wang *et al.* 2003a and is the bidding algorithm. It is implemented in much the same way as the previous two algorithms, but with a slight variation introduced to optimise energy consumption. It was deemed appropriate to analyse the algorithm's error-sensitivity from two directions, designated as Variation I and Variation II below. Details will follow in the next two subsections.

4.2.2.1 Bidding Algorithm - Variation I

The bidding protocol is an extension to the movement-assisted algorithms of VEC and VOR. The algorithm uses the mobile nodes to solve the coverage problems of the static nodes within the network. The static nodes use Voronoi polygons to determine the existence of local coverage holes. On discovery of a hole, the hole is weighted by some function on the size of the hole. The nature of the weight is pre-determined, but may simply be the area of the hole. The value of the weight is used by the mobile nodes to determine whether or not to fulfil the request by the static node. The higher the weight the higher the priority to cover the coverage hole – the process of determining the priority is known as bidding.

The algorithm executed by each mobile node consists an initialisation phase, followed by iterations through three phases: *service advertisement*; *bidding;* and *serving*. During the initialisation phase of the algorithm, the mobile nodes set the base threshold value to zero. Each mobile node broadcasts the threshold value to the static nodes, during the service advertisement phase. During the bidding phase, the static nodes calculate the size of local coverage holes. The area of the hole is measured and assigned as the node's bidding value. Static nodes send these bids as requests to the mobile nodes to move and solve the local holes. In the case of two or more mobile nodes responding to the static nodes call, the mobile node closest to the requesting node is used. The bidding value of the static node is assigned to the threshold value of the mobile node that partially or completely covers its local hole. The new position of the mobile node is determined as the furthest vertex in relation to the static node. This is known as the serving phase. The algorithm runs iteratively until no static node exceeds the mobile nodes threshold, i.e. until no weight of a static node is higher than the ones already healed by the mobile nodes

A potential problem to the algorithm is that of *duplicate healing* - Figure 4-6. Duplicate healing occurs when two static sensors, S_1 and S_2 detect a coverage hole and broadcast two independent requests for mobile sensors, M_1 and M_2, to solve the same hole at point *P*. Wang *et al.* (2003a) solve the problem by allowing one of the mobile sensors to reset its base threshold to zero. Thus, after the move is made, the next iteration will see the mobile node solve another coverage hole.

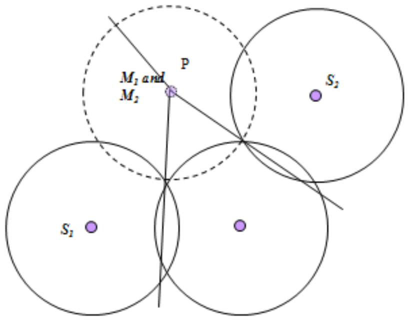


Figure 4-6 - Duplicate healing at *P* by S_c and S_d (Wang *et al. 2003a*)

Table 4-4 - Bidding Algorithm pseudo-code.

```
Notation:
  bid(loc, size) : target location of a mobile sensor, and the estimated additional
  coverage needed to heal a coverage hole
  $list_b$ : list of bid(loc, size)
  mobile(id, loc, base_price) : the id, location and base_price of a mobile sensor
  $list_m$ : list of mobile(id, loc, base_price)
  static(id, loc) : the id and location of a static sensor
  $list_s$ : list of static(id, loc)

Algorithm:
  At static node $N_i$
  1. Initialization
        a. Broadcast static locations
  2. Enter bidding phase
        a. Construct Voronoi polygons with $list_s$ and $list_m$ whose base price > 0
        b. Calculate the existence of a coverage hole
              i. If true then calculate the bid price of the hole
             ii. Find the closest mobile node $N_j$ from $list_m$ such that base_price
                 < h_size
            iii. Send bid to $N_j$ if $N_j$ is found
  3. Upon receiving mobile(j, $loc_j$, $base\_price_j$)  from $N_j$
        a. Add mobile(j, $loc_j$, $base\_price_j$) to $list_b$
  4. Upon receiving static(j, $loc_j$) from $N_j$
        a. Add static(j, $loc_j$)  to $list_s$

  At mobile node $N_i$
  1. Initialization
        a. Set $base\_price_i$ to 0
  2. Upon entering service-advertisement phase
        a. Broadcast mobile(i, $loc_i$, $base\_price_i$)
  3. Upon entering bidding phase
```

```
        a. If base_price ≠ 0
            i. Construct Voronoi polygons with $list_s$ and $list_m$ whose $base\ price > 0$
            ii. Calculate the existence of a coverage hole
                1. If true then calculate the bid price of the hole
                2. Find the closest mobile node $N_j$ from $list_m$ such that $base\_price < h\_size$
                3. Send bid to $N_j$ if $N_j$ is found
    4. Upon entering serving phase
        a. Check if $list_b$ contains nodes
            i. If true
                1. search $list_b$ for the node with the greatest coverage hole
                2. move $N_i$ to $h\_loc$, where $h\_loc$ is the destination for the node to heal the coverage hole
                3. set the base price of the $N_i$ to the size of the hole it heals
            ii. else
                1. do duplicate healing detection, setting $N_i$ $base\_price = 0$ if duplicate healing happens
                2. do local adjustment if no duplicate healing happens
    5. Upon receiving $mobile(j, loc_j, base\_price_j)$ from $N_j$
        a. Add $mobile(j, loc_j, base\_price_j)$ to $list_m$
    6. Upon receiving $static(j, loc_j)$ from $N_j$
        a. Add $static(j, loc_j)$ to $list_s$
```

For the first variation of the algorithm, inaccuracy was introduced to the static sensors only. The mobile devices are deemed to be accurate in their location placement. In other words the mobile nodes are accurately placed at the recommended position given by the static nodes.

4.2.2.2 Bidding Algorithm - Variation II.

The second variation of the bidding algorithm implements the algorithm in the same manner as before. However this time the location inaccuracy is placed on the mobile nodes. The static nodes will calculate the healing position accurately with the mobile nodes fulfilling this request, but with their resulting positions having some built in inaccuracy.

4.2.3 Static Algorithms

For the static algorithms we introduced more nodes into the ROI. By doubling the number of nodes within the network we introduce redundant nodes and overlapping of sensing areas – Figure 4-1 (b). The sensing radius, coverage radius and size of the ROI remain the same.

The Coverage-Preserving Node Scheduling Scheme and Optimal Geographical Density control algorithms, developed by Tian *et al.* 2002 and Zhang and Hou (2003) respectively, was chosen as the algorithms representing static networks.

4.2.3.1 Coverage-Preserving Node Scheduling Scheme (CPNSS)

The behaviour of static networks is different to that of mobile and hybrid networks in that nodes are unable to move. Instead of movement the networks use signal strength to reconfigure. On the detection of a coverage hole, the nodes increase their signal strength to provide sensor coverage. Tian *et al.* 2002 use node scheduling to reduce the energy consumption of sensors within large networks. Node scheduling is the process of switching nodes on and off, increasing the sensing radius of the active nodes to cover the entire network whilst turning off redundant nodes. By reducing the energy consumption of the nodes in the network, the entire lifetime of the network can be extended.

This algorithm is not a direct solution to solving coverage holes but instead is used to minimise energy consumption of the network. However the algorithm does use coverage-hole detection during the process of node scheduling. By detecting and solving coverage holes the authors guarantee that during the scheduling process coverage is addressed as well as possible.

The node-scheduling problem can be described by two problem statements. The first asks under what rule should the nodes in the network power down. In other words what makes a node redundant? This rule is known in this algorithm as the *coverage-based off-duty eligibility rule*. The second problem addresses the question of when and how often the nodes should reassess the network for redundant nodes.

Tian *et al.* 2002 address the first problem by calculating each nodes sensing area and then comparing it with its neighbours. This process is referred to as *sponsored coverage calculation.*

In this algorithm the set of neighbours, N(*i*), of a currently considered node, *i*, is defined as the set of nodes whose distance from the current node *i* is equal to or less than the sensing range, S_{ri}, of the current node. This is represented in Equation 4-2, where n denotes all the nodes deployed within the region of interest and *d(i,j)* is the distance between the current node *i* and the neighbour *j*.

Equation 4-2

$$N(i) = \{n \mid d(i\ j) \leq\ S_r\ n \neq i\}$$

Thus for current node *i*, the off-duty eligibility rule, Equation 4-3, can be represented as the union of the neighbours' sensing areas as a complete superset of *i*'s sensing area.

Equation 4-3

$$\bigcup_{j\, \epsilon\, N(i)} S(j) \supseteq S(i)$$

Figure 4-7 shows the rule applied to S_i. S_i is redundant in the sensing area of its three neighbouring nodes. As the GIS system is able to compute the union and intersection of polygons as well as the distance between special object, this case nodes, makes it easy to apply the eligibility rule. Thus Equation 4-3 is easily calculated using pre-existing GIS functionality.

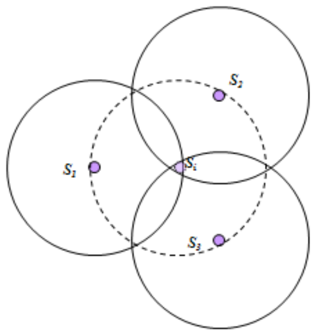


Figure 4-7 - Coverage of node S_i by neighbouring nodes (Tian *et al.* 2002).

Tian *et al.* 2002 proposes a two step iterative algorithm called the *coverage-preserving node scheduling scheme*.

The first step gathers "neighbourhood" information. Each node broadcasts its location to a distance equal to its sensing radius. Nodes that receive a message from this signal are set as the broadcasting nodes neighbours.

Post collection of neighbour information, each node evaluates its eligibility to power off based on Equation 4-3. If all nodes decide simultaneously to turn off, coverage holes may occur. Thus the authors introduce the back-off scheme where each sensor determines its power down eligibility after a random time, T_d. When a node decides to power down it sends a message to its neighbouring nodes. The neighbouring nodes then remove the eligible node from their list of nodes. Nodes that have a long T_d will not consider the nodes that have already powered down.

The sensor remains in a hibernation state until a 'power on' broadcast has been received or a so-called hibernation timeout, T_w elapses after a fixed period.

The complete process is described in Table 4-5.

Table 4-5 – CPNSS pseudo-code.

```
Notation:
$S_i$ Current node
$N_i$ Neighbouring nodes
$T_d$ Time delay for nodes to calculate their power down eligibility
$T_w$ Time that the sensors remain in hibernation

Algorithm:
  At static node $S_i$
  1. Neighbour Information Obtaining Step
       a. Broadcast location a distance equal to $S_i$ sensing range
       b. Receive neighbours broadcasted signal, compiling a list of neighbours
          $N_i$
  2. Wait for random eligibility time $T_d$
  3. Assess for power down eligibility, insuring that the nodes have sufficient
     power to stay active during $T_w$
       a. If true
            i. Broadcast a power down message to all neighbours
           ii. Go into hibernate state
  4. Hibernate until a power up message has been received or hibernation timeout
     has occurred, $T_w$.
```

4.2.3.2 Optimal Geographical Density Control (OGDC)

The OGDC presented by Zhang and Hou (2003) shares similarities with the CPNSS algorithm presented above, in that they both run iteratively implementing a form of node scheduling. OGDC however makes three key assumptions:

- The radio coverage of each sensor is at least twice the distance provided by sensor coverage. As discussed in Zhang and Hou (2003), this assumption implies complete connectivity to a completely covered ROI.
- Nodes are aware of their own location.
- Nodes are controlled via some form of time synchronicity.

As opposed to CPNSS, OGDC allows nodes to be in any one of three states namely UNDECIDED, ON or OFF. The algorithm runs iteratively, dividing the lifetime of the network into rounds. At the beginning of each round all nodes within the network are set to UNDECIDED. A random node S_i is chosen to be a starting point for determining the working nodes, where working nodes are defined to be nodes in the ON state. A neighbour, S_j of S_i is chosen. The neighbour is a random node selected from a group of nodes within a distance of $\sqrt{3r}$ to $S_{i,}$, where r is the sensing radius. S_j is set to ON and all other nodes in S_i's sensing radius are set to OFF. The next optimal position for a node is computed, denoted as O in Figure 4-8. This optimal position is on the line that bisects the line connecting S_i and S_j. It is at a distance of r from the line connecting S_i and S_j. (There are, of course, two optimal positions - one on either side of the bisected line.) The next node to set ON, S_k, is the one closest to an optimal position. The cycle is repeated with the neighbour node of S_k being selected.

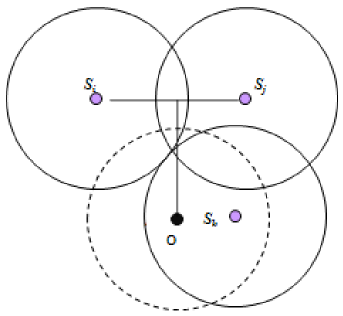


Figure 4-8 - Determining the neighbouring node of S_i.

Table 4-6 – OGDC pseudo-code.

Notation: S_x Current sensor T_d Time to wait before repeating the algorithm

```
Algorithm:
  1. Set all nodes to UNDECIDED
  2. Select node $S_i$ randomly from set of nodes
       a. Set $S_i$ to ON
  3. Determine the neighbouring node of $S_i$ as $S_j$
       a. Select all nodes within a radius of $\sqrt{3r}$
       b. Randomly select node $S_j$ from this set
       c. Set to $S_j$ ON
       d. Set all other nodes in the set of neighbours as OFF
  4. Determine the next optimal position of a third node
       a. Calculate the ideal position of the third node by constructing a line
          from $S_i$ as $S_j$. A second line perpendicular to the first a length equal to
          the sensing radius.
       b. Repeat in the opposite direction to calculate the north bound sensor
  5. Select $S_k$
       a. Find the closest UNDECIDED node to the optimal position
       b. Set the closest node as $S_k$
  6. Repeat the algorithm from step 3 by determining the neighbour of $S_k$
  7. Terminate the selection of Working nodes once all nodes are set to ON or OFF
  8. Allow the algorithm to execute for time $T_d$ where $T_d$ is determined as a time
     greater than that needed to select the working nodes, but less than that of the
     lifetime of the sensor with the least lifespan from the list of working nodes
```

4.3 Chapter Conclusion

In this chapter we discussed each of the five algorithms as described by their original authors. Two were mobile algorithms, one was a hybrid, and another two where static algorithms. Algorithm selection for this study was based on the results obtained by the original developers, i.e. algorithms that performed reconfiguration with the least effort were chosen. A discussion on each of the experiments to be performed was also given, outlining the methods used to assess the impact that location inaccuracies might have. The following chapter discusses the results of the experiments that were carried out to assess the impact location inaccuracy might have on the five algorithms.

Chapter 5

Analysis and Results

"The time with which we have to deal
is of the order of two billion years.…
Given so much time the 'impossible' becomes possible,
the possible probable, and the probable virtually certain.
One has only to wait: time itself performs miracles."

G. Wald, "The Origin of Life," - 1955

Based on the experiments outlined in the previous chapter, the present chapter addresses the results and findings of the experiments. The two algorithms from each deployment type are compared to each other followed by a comparative conclusion.

5.1 Mobile Algorithms

The following section examines the results obtained for the two movement-assisted algorithms. During the control experiments, a limit on the mean total distance travelled by the nodes served as a termination condition for the iterations: after each iteration, the distance travelled by each node from its *initial* to its present position, was summed over all nodes and divided by the number of nodes. If this mean distance travelled is greater than the limit, then the algorithm is terminated; otherwise another iteration is executed.

A mean total distance of 10 meters was chosen as a terminating condition for both algorithms. This corresponds to the communication range that had been assigned to the nodes.

5.1.1 VECtor-based Algorithm (VEC)

The VEC algorithm was assessed using the experiments outlined in the previous chapter. The control experiment enforced a terminating criterion of 10 meters mean total distance. The criterion resulted in four iterations of the algorithm. Figure 5-1 represents each of the iterations. The original coverage hole was calculated as 31.57% (68.43% of the ROI is covered). After the fourth iteration the coverage hole had been reduced to 16.21% of the ROI, and 83.79% of the ROI was now covered. This means that the coverage hole was reduced by approximately 49%

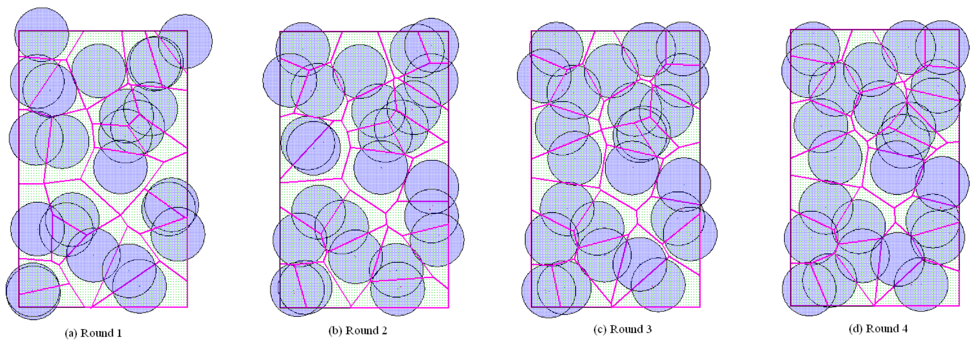


Figure 5-1 - Execution of the VEC Algorithm over four iterations.

The above experiment correlates with that of the original authors, Wang *et al.* (2004). The results provide a level of confidence that the algorithm was implemented accurately.

As a single starting scenario is used in the control experiment, it could be said that the placement of the nodes assist in the results obtained. To determine if the starting scenarios of the nodes play a role in the overall outcome of the algorithm the following experiment was setup. Fifteen scenarios were configured. In each case, the starting position of each node was randomly determined. The coverage holes ranged between 32.5% to 44.7%. As oppose to using the 10-meter terminating condition, the algorithm was run for 100 iterations for each of the 15 starting scenarios. This means that nodes were allowed to drift as far from their original position as was dictated by 100 iterations of the algorithm.

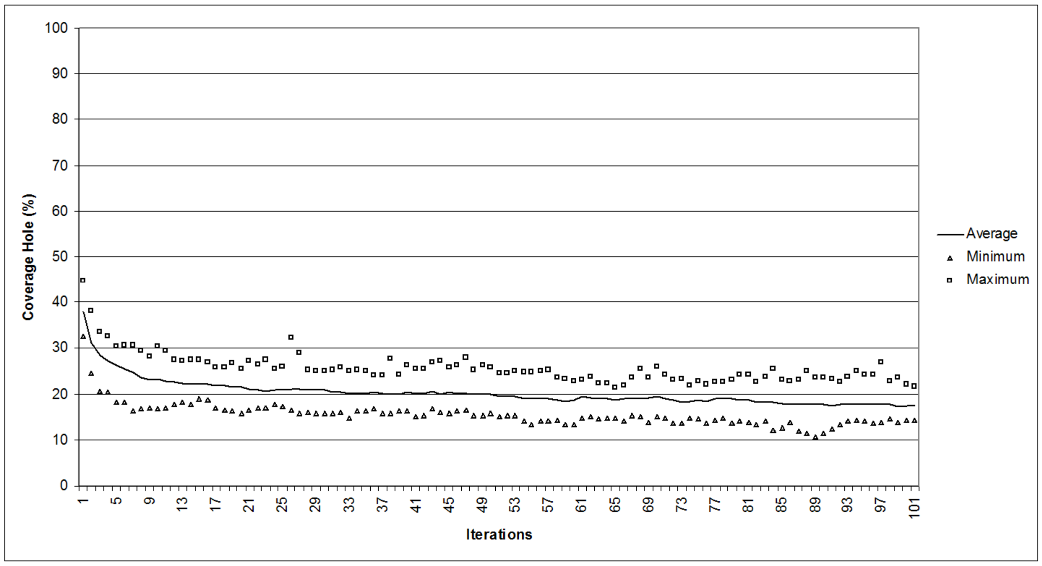


Figure 5-2 - Alternative randomly deployed starting scenarios for the VEC algorithm.

The results of variable starting positions for the VEC algorithm are shown in Figure 5-2. The graph displays – for the 15 starting scenarios – the average, maximum and minimum coverage hole size at each iteration. All initial scenarios generally improve over the 100 iterations, the final average, maximum and minimum coverage hole percentages being 17.53%, 21.68% and 14.4% respectively. Although improvement is not guaranteed from one iteration to the next, no starting scenario causes the VEC algorithm to diverge as the number of iterations increased. After approximately 20 iterations the average improvement is approximately 20% and its rate of decline slows down considerably. At about 90 iterations, the minimum coverage hole of about 10% is attained.

Using the statistical observation we see that the values of coverage hole percentage observed (Observed Value percentage axis on the graph) at the end of each scenario falls within the upper and lower outlier limits. This result affirms that the outcomes of the various scenarios are not significantly different - Figure 5-3. Because of this it was assumed that the starting scenario used in the control experiment could be used to adequately represent results that were obtained by other scenarios if they were to be run. As a result, the starting scenario in the control experiment was used in subsequent experiments described below.

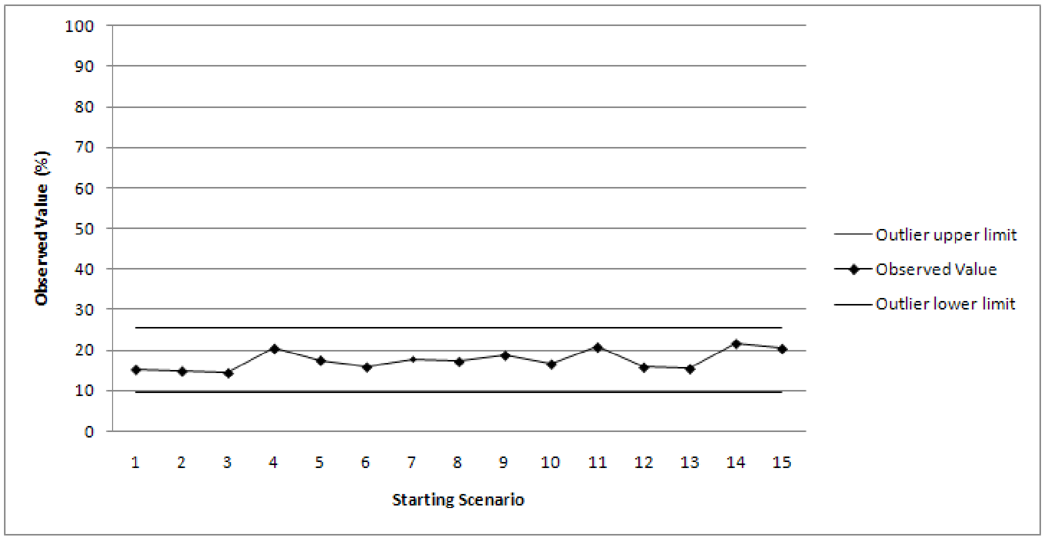


Figure 5-3 - Using outliers to compare the resulting coverage hole of each starting scenario (VEC).

The simulations of the control experiment assumed a best-case scenario – one in which all the nodes in the network accurately reported their position to the same extent. The initial deployment setup of the control experiment was used to run each algorithm with various levels of inaccuracy. Twenty five simulations were run per algorithm. In each case, gradual increases in the number of inaccurate nodes were set to determine the VEC algorithm's degree of recovery as the number of accurate nodes increases. When there are no accurate nodes, the coverage hole varies between about 23% and 37%, depending on the level of inaccuracy. When 20 out of the 25 nodes report accurately, at 2 meters inaccuracy the coverage hole drops to around 20%, while at 4, 6 and 8 meters inaccuracy the final recovery is the same at about 26%. The slope of graph indicates a gradual increase in coverage hole per new inaccurate node introduced in a somewhat linear fashion. Very broadly, one could say that each inaccurate node decreases the VEC algorithm's effectiveness.

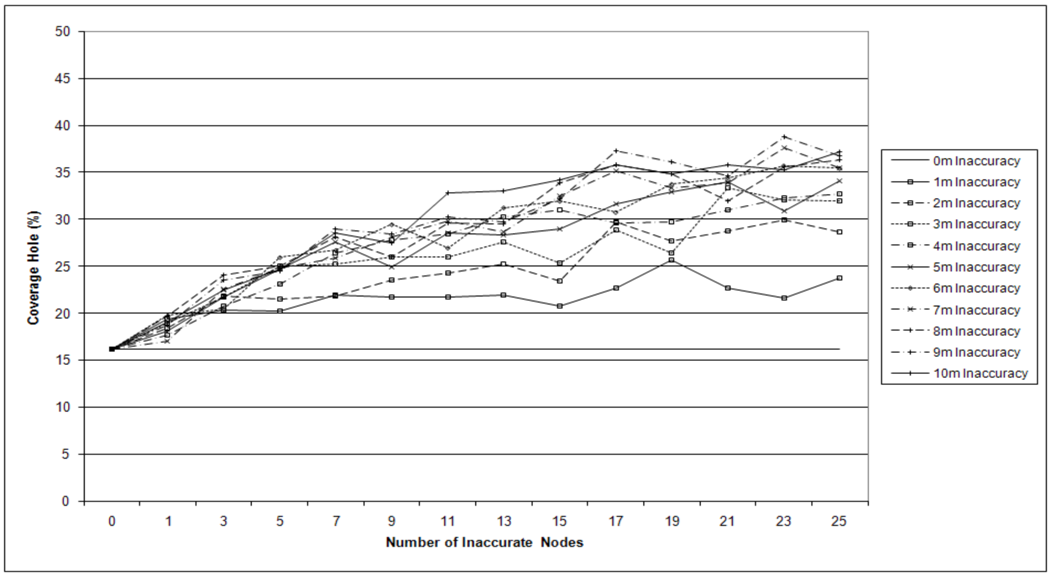


Figure 5-4 - Increasing the percentage of inaccurate nodes (VEC).

The previous experiment shows a gradual increase in inaccuracy as the number of inaccurate nodes increased, the following experiment assesses the algorithm from a worst-case scenario i.e. all the nodes are assumed to be inaccurate.

Twenty simulations were run per scenario. In each case, *all the nodes* were set at an inaccuracy level of *n* meters, where the inaccuracy, *n*, ranged from 1 to 10 meters, increasing in 0.5 meter intervals. The termination criterion used in the control experiment was retained. For each inaccuracy level, say of *n*, the algorithm was run 50 times. Each such run located each node in each iteration at a displacement of *n* meters away (in a randomly determined direction) from the node's previous position as determined by the VEC algorithm. The minimum, maximum and average coverage holes were computed over these 50 runs.

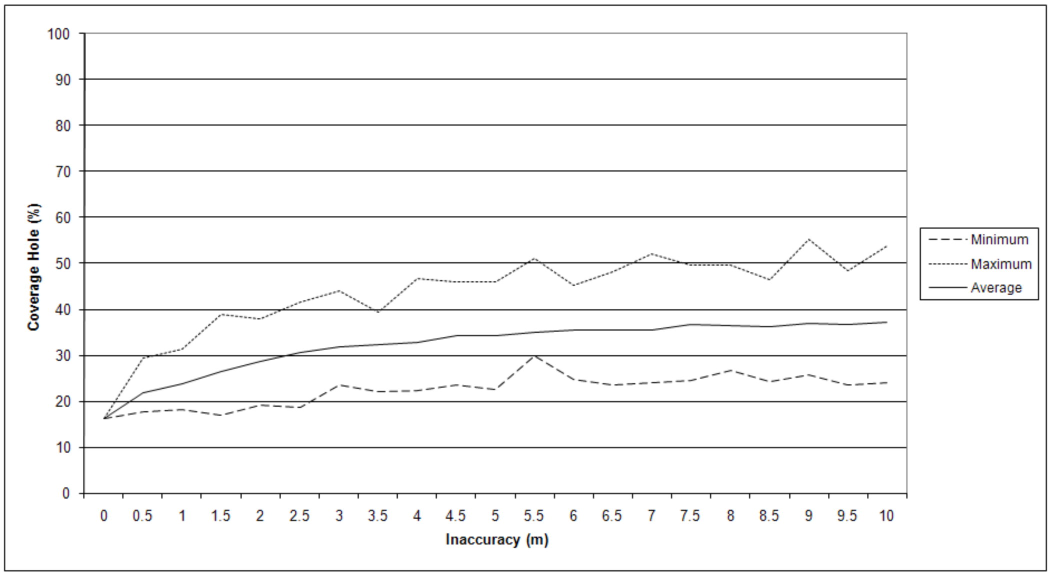


Figure 5-5 - Coverage hole(%) related to an increase in location inaccuracy (VEC).

Figure 5-5 shows the results obtained for the VEC algorithm. The coverage hole rapidly increases as the inaccuracy level is increased – i.e. inaccuracy has a pronounced effect on coverage. For example, at an inaccuracy of 2.5m the initial accurately-determined coverage hole of 16.20% almost doubles to 30.68% on average. As inaccuracies increase to 5 meters per node, the coverage hole grows to about 35%.

Interestingly, the coverage hole remains at around 37% for higher inaccuracies. This is evidently due to the way in which the VEC algorithm deals with the ROI boundary. If the node determines its new location to be outside the boundary, then the VEC algorithm does not allow the node to move.

The foregoing described experiments that were based on a termination criterion applied to the VEC algorithm, whereby the total average node movement relative to original node position was limited to 10 meters. It seemed important to verify that this did not represent some artificial termination point, and that significant coverage improvement could not perhaps be gained, even in the face of inaccuracies, by increasing the number of iterations. In Figure 5-6, the coverage is shown under the various worst-case inaccuracy scenarios as the number of iterations was increased to 100.

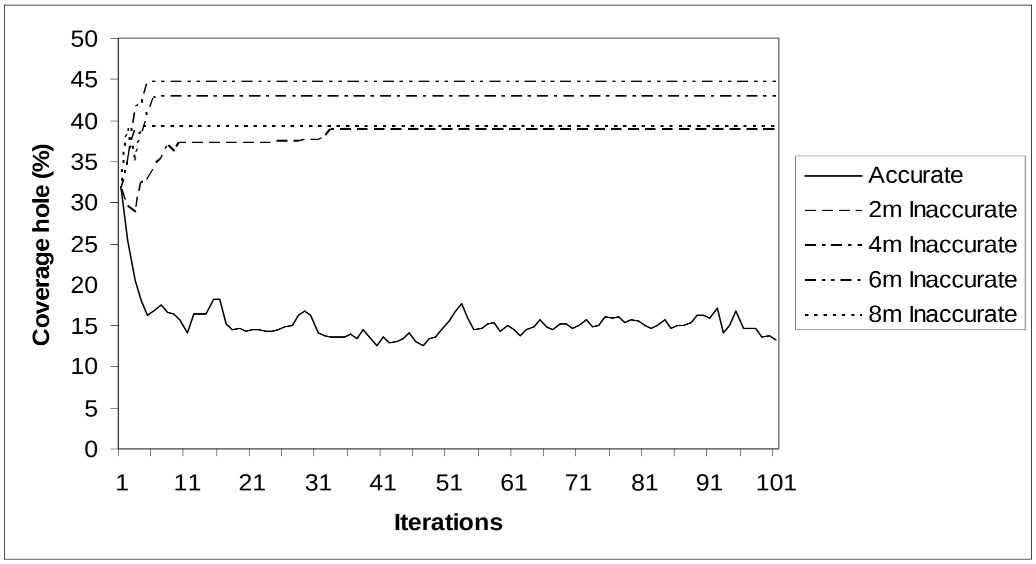


Figure 5-6 - Coverage hole (%) by number of iterations (VEC)

In Figure 5-6 it is evident that in all cases, no significant gains are to be had by increasing the number of iterations. For example, the simulation cases for the accurate locations shows a 16.21% coverage hole after four iterations and 13.22% after 100. In the presence of any form of inaccuracy, the algorithm diverges, and all nodes eventually drift to the boundary.

5.1.2 VORonoi-based Algorithm (VOR)

The same terminating criterion for the control experiment was implemented during the execution of VOR. As with VEC, this resulted in four iterations. Figure 5-7 shows the results of VOR after each iteration. After the fourth iteration the coverage hole had been reduced to 14.12% of the ROI, and 85.88% of the ROI was now covered. In comparison to the VEC algorithm, the coverage hole reduction was slightly larger – approximately 55%.

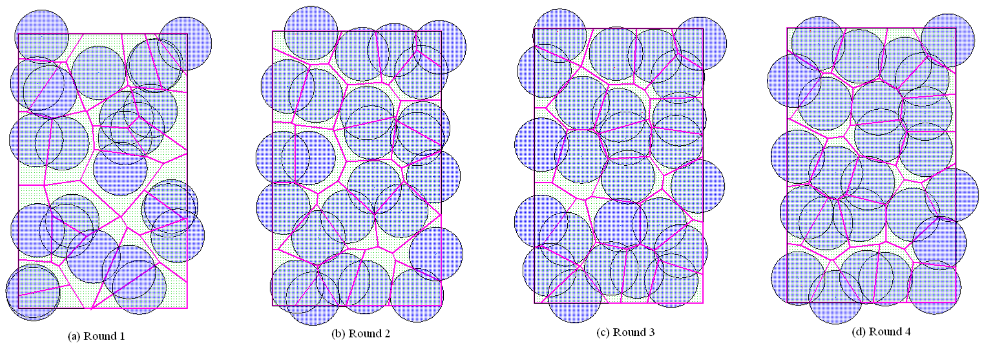


Figure 5-7 - Execution of the VOR Algorithm over four iterations.

The final result shows a network with significant improvement in coverage over the ROI.

Figure 5-8 shows the results for the VOR algorithm for various random node deployment as starting scenarios for the VOR algorithm. The overall picture is similar to that provided by the VEC algorithm, although convergence (below 10% coverage after 15 iterations) and overall performance is somewhat better. The final average, maximum and minimum coverage hole percentage is 8.6, 12.4 and 5.5 respectively, and a minimum coverage of less than 5% is attained several times after 80 iterations.

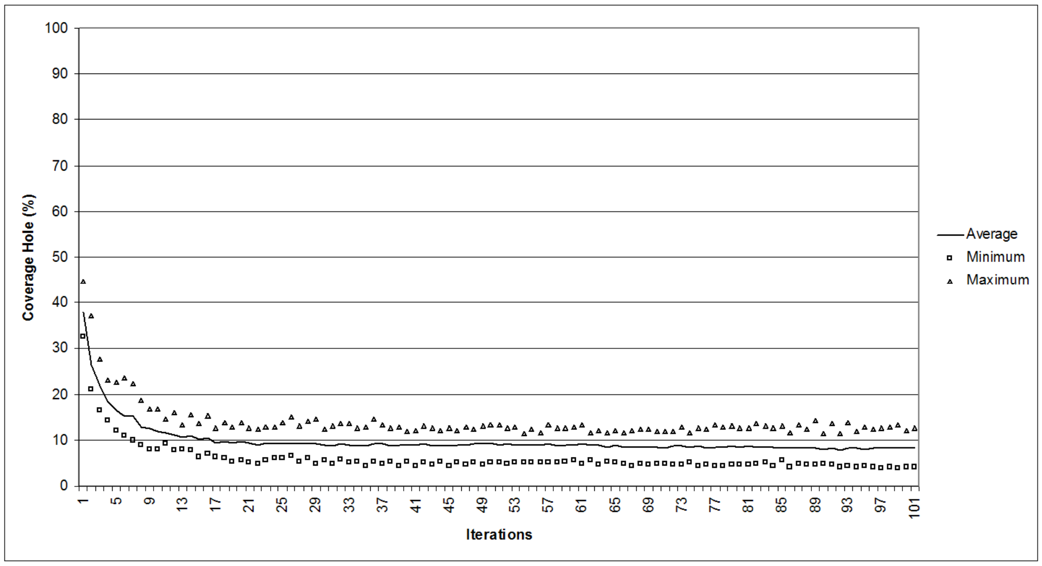


Figure 5-8 - Alternative randomly deployed starting scenarios for the VOR algorithm.

As can be seen in Figure 5-9 the upper and lower outlier limits are less than those of the VEC algorithm. This is due to the lower levels of coverage holes resulting in lower first and third quartiles. The observed coverage hole (Observed Value percentage axis on the graph) values still lie between the upper and lower limits suggesting that they are not significantly different from one another. As with VEC, it can be said the control starting scenario is a fair comparison for the behaviour of the algorithm for the forthcoming experiments.

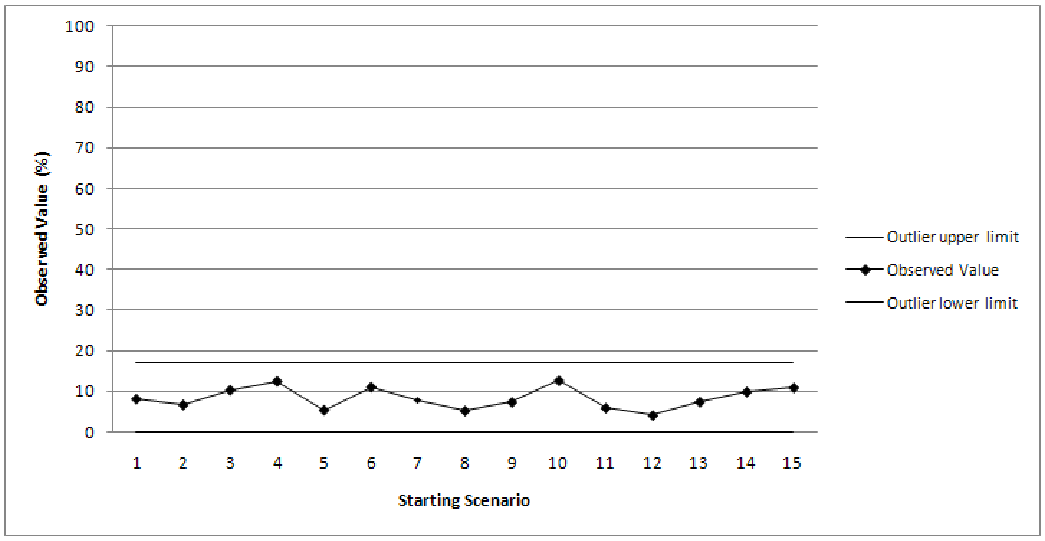


Figure 5-9 - Using outliers to compare the resulting coverage hole of each starting scenario (VOR).

As can be seen in Figure 5-10, the VOR algorithm generally performs better in the presence of inaccuracies than the previous VEC algorithm. The VOR algorithm seems reasonably tolerant of inaccuracies, even fairly large ones, provided that the number of inaccurate nodes is limited. By "fairly large" it is meant, inaccuracies (6 to 8 meters) that are of the same order of magnitude as the sensing radius (5 meters). When 5 nodes were inaccurate at a level of 8 meters (i.e. 20% of the 25 nodes were inaccurate), the coverage hole was about 20% of the ROI – a mere 6% degeneration from the control coverage hole of 14.12%. On the other hand, at 2 and 4 meters inaccuracy, the algorithm's performance appears to be reasonably indifferent to the number of accurate nodes, suggesting that VOR is quite robust in the presence of relatively small inaccuracies. In these cases, the coverage hole remains close to the control of 14.12%, and indeed, in certain instances drops below it.

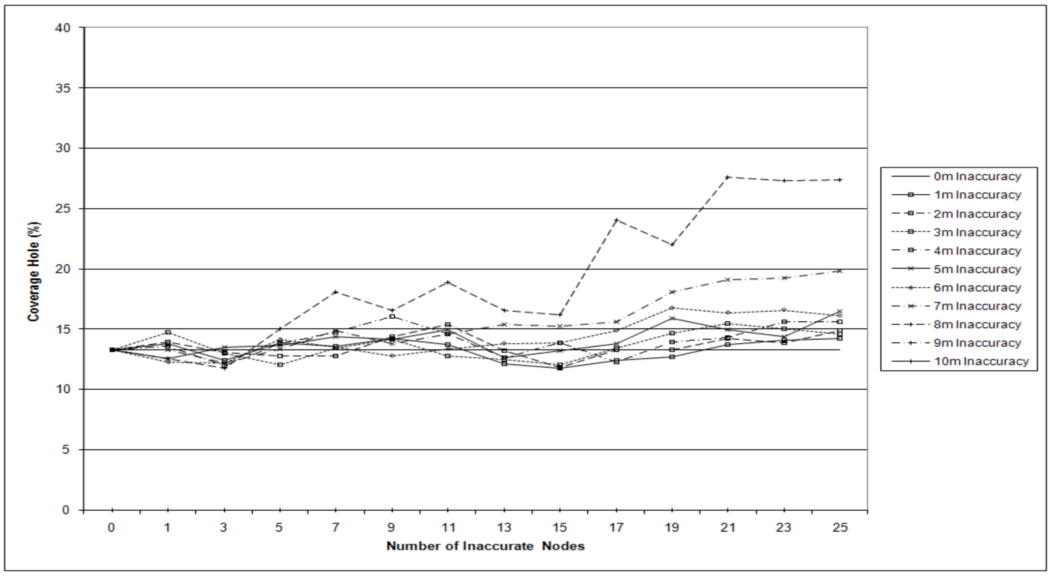


Figure 5-10 - Increasing the percentage of inaccurate nodes (VOR).

Figure 5-11 shows the results obtained for the VOR algorithm as location inaccuracy is increased. In this case, the coverage hole percentage recovery is relatively robust for inaccuracies up to about 6 meters. Thereafter, the inaccuracy has a relatively pronounced effect on coverage. As the location inaccuracy increases, so does the coverage hole. A coverage hole of approximately 15%, for an inaccuracy level of 6 meters, increases by more than 10% when the inaccuracy doubles to 8 meters. This is clearly due to the VOR algorithm's dependency on location both during the discovery phase when creating polygons, as well as during the movement phase in determining the position to which the node should move.

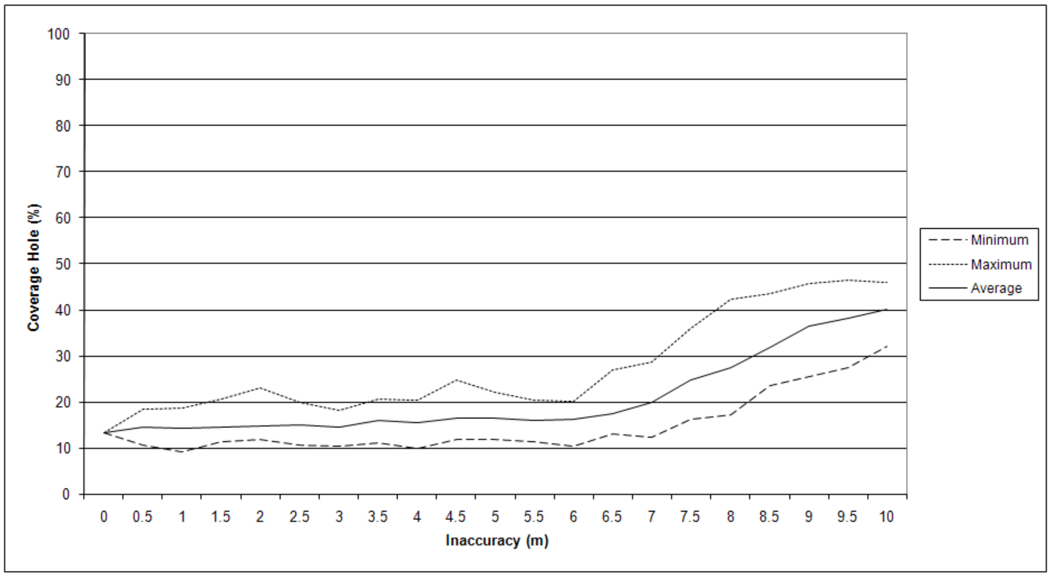


Figure 5-11 - Coverage hole (%) related to an increase in location inaccuracy (VOR).

To highlight how such a decline in effectiveness could occur, consider the scenario in Figure 5-12. During the discovery phase of the VOR algorithm, it constructs a local Voronoi polygon to determine the existence of a coverage hole. If the locations broadcast by the neighbouring nodes are inaccurate, then the local Voronoi polygon of S_i is erroneous. Figure 5-12 indicates that the accurate Voronoi polygon, S_i would result in a shift towards vertex v3. However, due to the inaccurate location details broadcast to S_i by neighbouring nodes, S_i could instead be moved in a totally different direction, presented in the figure by v5. As a result, in each iteration the nodes may move away from their optimal locations. Unlike the VEC algorithm, this also applies to the nodes on the boundaries of the ROI: if locations are inaccurate, then these nodes become unaware that the calculated movement could place them outside the ROI. The nodes knowledge of the boundary is limited to the edge of the Voronoi polygon.

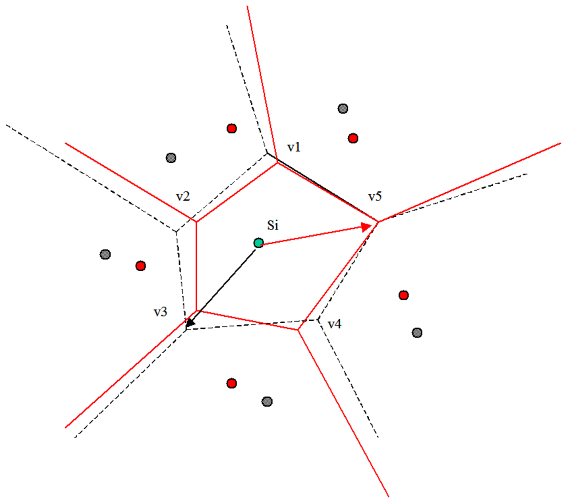


Figure 5-12 - Inaccurate calculation of a Voronoi polygon.

In Figure 5-13 it is evident that no significant gains are to be had by increasing the number of iterations. For example, in the case of the simulations based on accurate location data, the percentage of ROI that had coverage holes dropped from 14.12% after four iterations to 11.86% after 100 iterations. Indeed, the data shows that in the presence of high inaccuracy (8 meters) the algorithm diverges, and eventually all nodes drift out of the ROI.

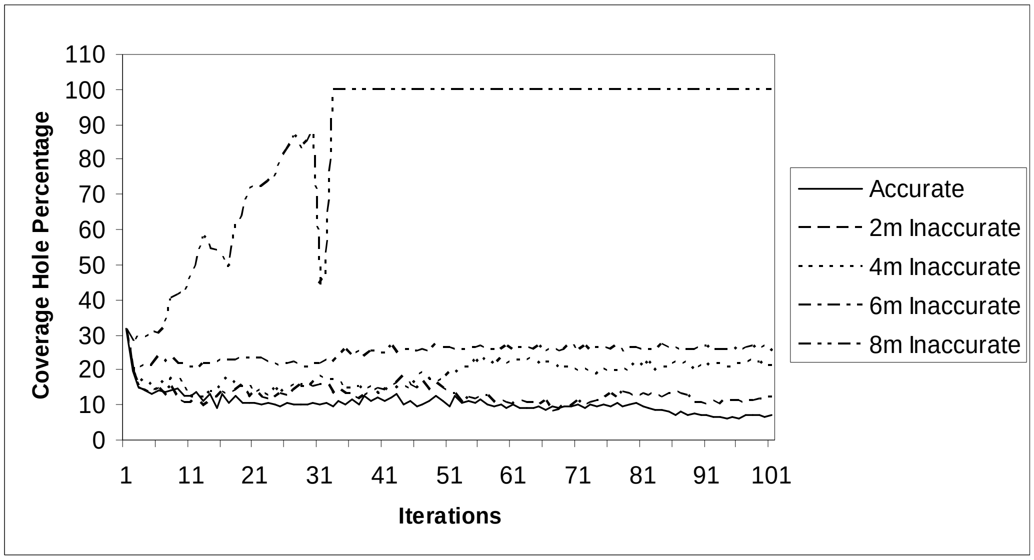


Figure 5-13 - Coverage hole (%) by number of iterations (VOR)

5.1.3 Conclusion

The above assessment of the two movement-assisted coverage algorithms suggests that the VOR algorithm is reasonably robust if the inaccuracies are somewhat lower than the sensing distance. It remains reasonably robust when the inaccuracies are somewhat higher, provided that they do not affect a very high proportion of nodes.

On the other hand, the VEC algorithm shows a high dependency on the accurate calculation of node locations. However the algorithm acquires a certain type of robustness in relation to node behaviour at the ROI boundary.

By keeping all the nodes within the ROI, in the high-inaccuracy experiment, nodes may cluster along the boundary, but a coverage hole of 100% can never occur.

5.2 Hybrid Algorithms

This section examines the behaviour of two hybrid algorithms. For the purpose of the control experiment, the terminating condition is when no movement of mobile nodes occurs in a complete iteration. It can be assumed that when no movement occurs, this is because the mobile sensors are unable to further heal the coverage hole of any static nodes.

The same network layout used in the mobile network topology is reused as the static network. This network layout results in 25 static nodes randomly placed within the ROI. A balance of 23 mobile nodes was used by the algorithm during execution. The balance of mobile nodes was determined as the number of nodes required to completely cover the coverage holes left by the static network. A more efficient method for calculating the ratio between mobile and static nodes is deemed outside the scope of this study and is to be considered for future research.

As hybrid networks are dependent on two forms of node deployment, namely static and mobile, this study deemed it to be appropriate to assess the effects on a popular algorithm of node location inaccuracy from the two corresponding view points.

5.2.1 Bidding Protocol – Variation I

For this variation of the algorithm, location inaccuracy is in reference to the positions of the static nodes contained in the network. Thus, the positions of the mobile nodes are computed by the static nodes based on inaccurate information of static node positioning. Thereafter, mobile nodes are *accurately* placed at the positions determined for them.

Figure 5-14 shows visually the results of algorithm. As can be seen not all the coverage holes were healed. The algorithm ran a total of 20 iterations (however no change to the network is seen after 7 iterations) with the final result of 4.1% coverage hole being achieved after the sixth iteration, and no subsequent improvement — see Figure 5-15. As the algorithm runs until no movement from a mobile node can resolve coverage holes greater than the ones currently being resolved, the termination criterion is as a result of the execution of the algorithm. For this reason the algorithm determines the maximum number of iterations that may be executed and at best the simulator may be set to reduce that number. For the assessment of this and the next variation the simulator allowed the algorithm to execute the maximum number of times. The algorithm did not behave as efficiently as expected. There is a significant improvement to the network coverage but the overall network coverage after the algorithm is completed could still be improved.

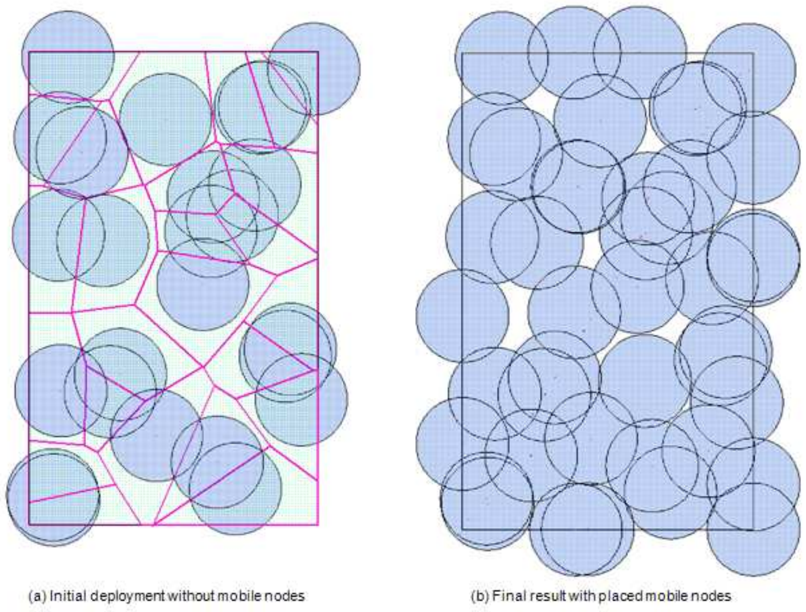

(a) Initial deployment without mobile nodes
(b) Final result with placed mobile nodes

Figure 5-14 - Results of the bidding algorithm after the control experiment.

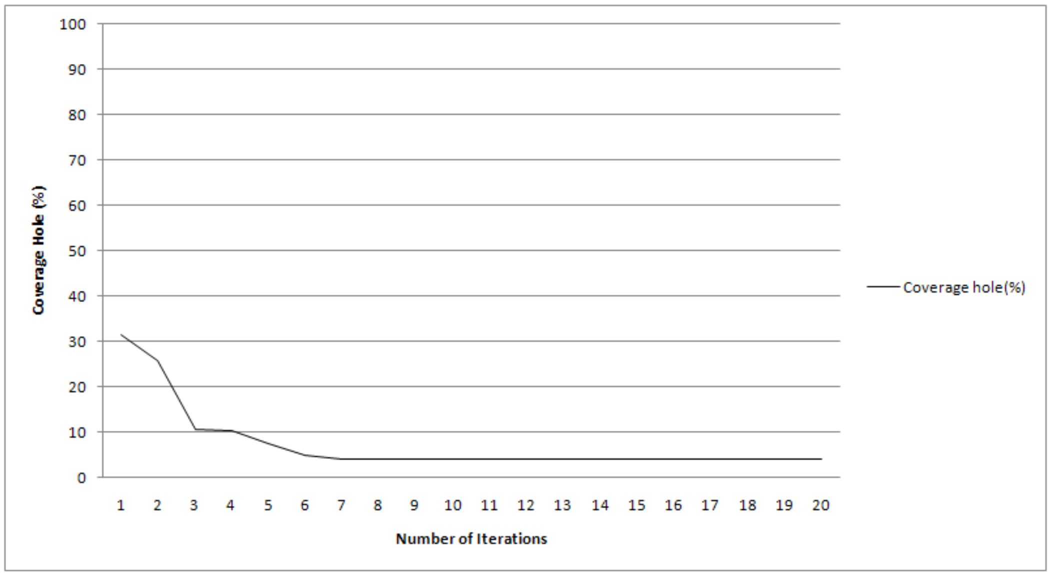


Figure 5-15 - Coverage hole results of the control experiment.

The sum of redundant nodes placed in the network as mobile nodes to solve the coverage holes should have allowed the network to completely cover the ROI.

A reference to the next figure shows two scenarios that illustrate why the algorithm cannot guarantee full coverage, even if the total mobile node coverage should, in theory, be sufficient to cover all holes.

Figure 5-16(a) shows a mobile node, P_1, placed at the furthest vertex to solve a coverage hole. Since this new position lies at the edge of the network, three quarters of the node's sensing ability is wasted on an area outside the ROI.

Figure 5-16(b) illustrates the notion of what this study refers to as close proximity healing. This is where the vertices to which the mobile nodes P_2 and P_3 are moved, are very close to each other. When close proximity occurs the nodes waste large sums of sensing ability due to overlapping with other nodes. Close proximity healing is similar to duplicate healing—i.e. a scenario in which two mobile nodes are move to the same vertex. However, the algorithm explicitly avoids duplicate healing, but does not disallow close proximity healing.

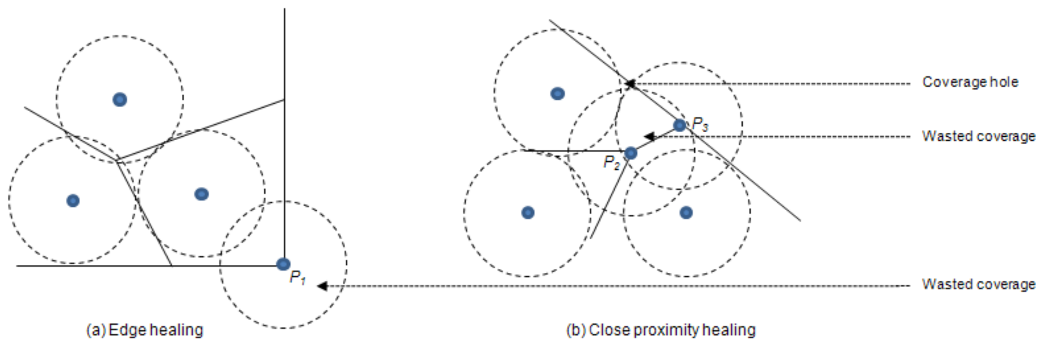


Figure 5-16 - Flaws with the bidding algorithm.

A further problem that may occur is that after all the mobile nodes are placed, small coverage holes – shown in Figure 5-16(b) – may still exist. These holes are never solved as the bidding value assigned to the mobile nodes for the previous holes solved are greater than the smaller coverage hole. As shown in the Section 4.2.2 the mobile nodes will only move if the coverage hole to solve is greater than the current hole it is covering.

No doubt, the algorithm could be optimized by, for example, shifting nodes slightly away from the vertices when close proximity healing occurs. Optimisation of the algorithm is to be considered as a future research opportunity.

To check for consistency, the same mobile and static nodes were deployed from fifteen randomly selected starting scenarios.

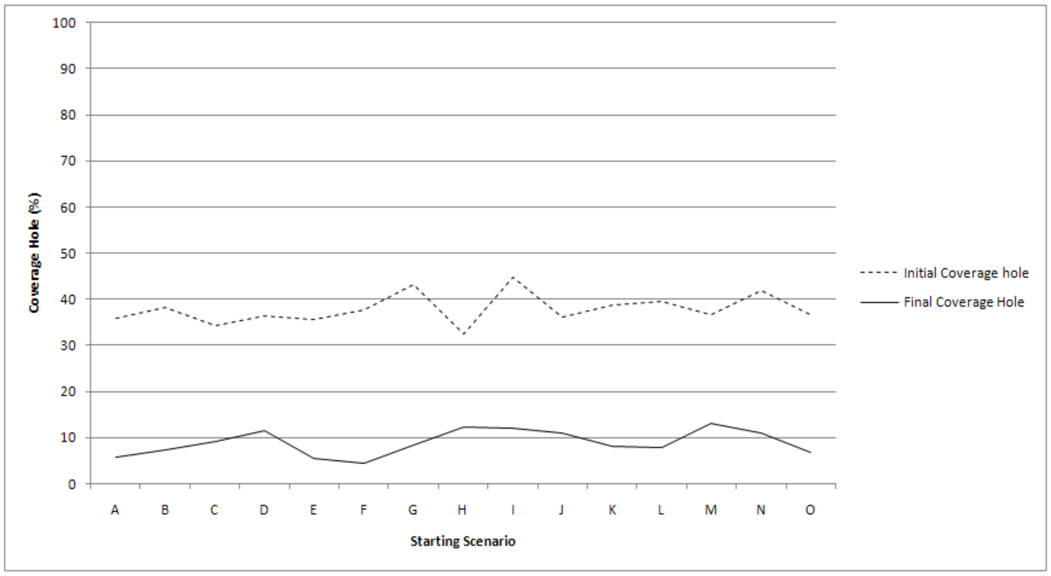


Figure 5-17 - Alternative randomly deployed starting scenarios for the Bidding algorithm.

As with the *control* experiment above, Figure 5-17 shows that there was a significant improvement to the network coverage for each of the 15 starting scenario. The algorithm could still be improved, since a 0% coverage hole is never achieved, even though there are sufficient nodes in the network to accomplish this. The figure shows the initial coverage hole percentage in the ROI excluding the mobile nodes. The average coverage hole percentage taken over the 15 starting scenarios is 37.8%. The figure also shows the final coverage hole percentage after the algorithm has been executed. Execution of the algorithm improved the starting scenarios by an average of 60%, where A, E and F improved slightly more than the rest. The algorithm converged after an average of 5 iterations over the 15 starting scenarios—much the same as that of the control experiment’s 6 iterations.

The results of the gradual increase in inaccurate static nodes within the network are shown in Figure 5-18. As can be seen, there is no obvious connection between the number of inaccurate nodes and coverage. In previous algorithms the corresponding graph shows a tendency for coverage performance to degrade as more nodes become inaccurate. In other words as the number of inaccurate nodes increase so does the overall coverage hole.

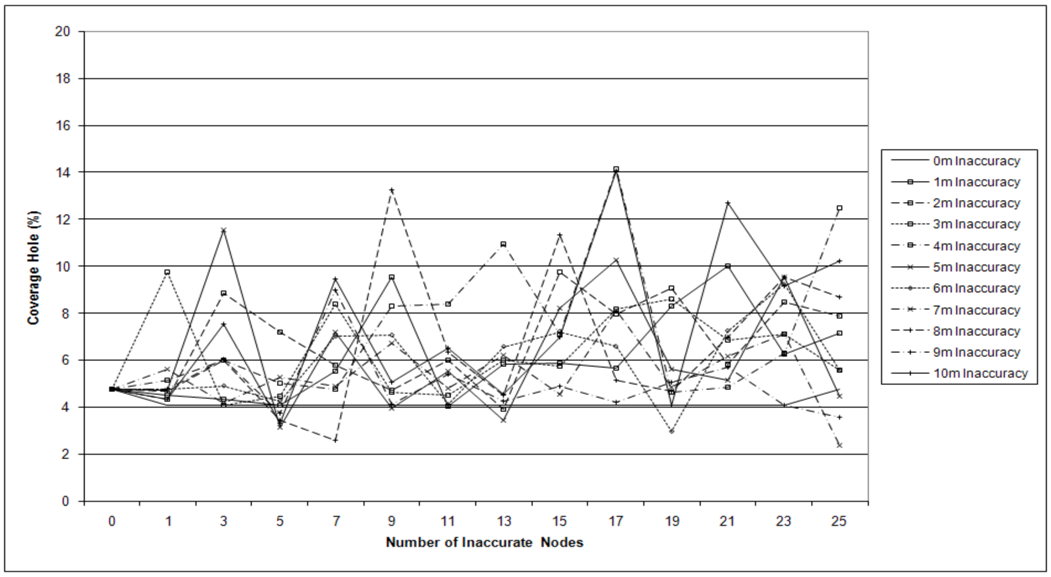


Figure 5-18 - Increasing the percentage of inaccurate nodes (Bidding Protocol I).

In contrast the above graph shows a very random result. This can also be seen in the next experiment where all static nodes are set be inaccurate.

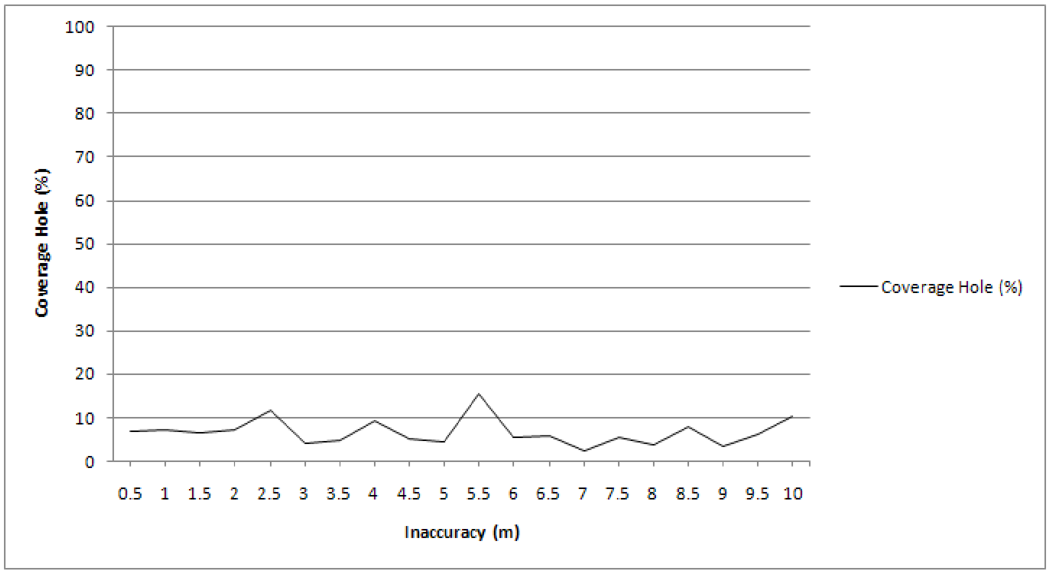


Figure 5-19 - Coverage hole(%) related to an increase in location inaccuracy (Bidding Protocol I).

Figure 5-19 shows the effects of complete inaccuracy injected into all the static nodes. The experiment is run using inaccurate location details between 0.5 and 10 meters. The inaccuracy is injected when the static nodes broadcast the location to the mobile nodes. The mobile nodes in this variation are placed without inaccuracy at the position suggested by the static node. The results obtained also show a random outcome.

As the static nodes are fixed in their position, it can be said the network would maintain at least the degree of coverage provided by the static nodes. The initial coverage hole within the ROI without the mobile sensors is 32%. The worst case coverage hole size in this experiment was 11.73%—more than double that of the control experiment. As the static nodes determine the placement of the mobile nodes, in a worst case scenario boundary static nodes may place some or all of the mobile nodes outside the ROI, resulting in the spikes seen in Figure 5-19. As not all the static nodes are on the boundary, a large percentage of the mobile nodes would still be kept within the ROI.

As the termination of the algorithm is determined by the mobile nodes in a reasonable time, the termination criterion experiment is omitted from this variation.

5.2.2 Bidding Protocol – Variation II

This second variation of the algorithm was executed in the same way as that of the first variation discussed in Section 5.2.1. The difference was that location inaccuracy is now placed within the mobile nodes. The static nodes determined the position at which for the mobile nodes should be placed. Once the mobile nodes are made aware of the new position, this position is injected with a degree of inaccuracy.

The *control* and *initial deployment sensitivity* experiments were omitted, as they would yield the same results as that of the first variation.

The results of the gradual increase in inaccurate mobile nodes within the network are shown in Figure 5-20. The coverage is only negatively affected by the inaccuracies when a large number of the mobile nodes are inaccurate. This is understandable as the static nodes make up a substantial degree of the ROI network coverage. The mobile nodes simply act as assistants to the static nodes. In a worst case scenario all of the mobile nodes could be placed outside the ROI resulting in no change in network coverage. Thus the network can always assume a worst case coverage of the same degree that the static nodes are providing in the initial network deployment.

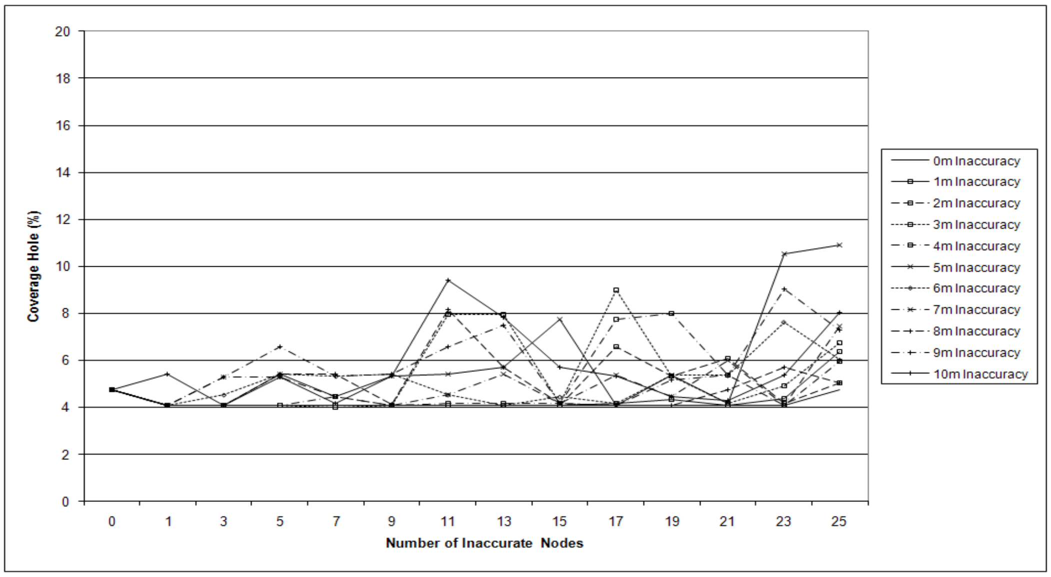


Figure 5-20 - Increasing the percentage of inaccurate nodes (Bidding Protocol II).

Different from the previous variation, the above graph shows an increase in coverage hole as the number of inaccurate nodes increases. A definite threshold at around 9 inaccurate nodes can be seen. At this point the algorithm begins to behave in much the same way as that of the previous variation in that the coverage hole is random with the same effect seen irrespective of the degree of inaccuracy. This randomness in the coverage hole is seen again in the next experiment.

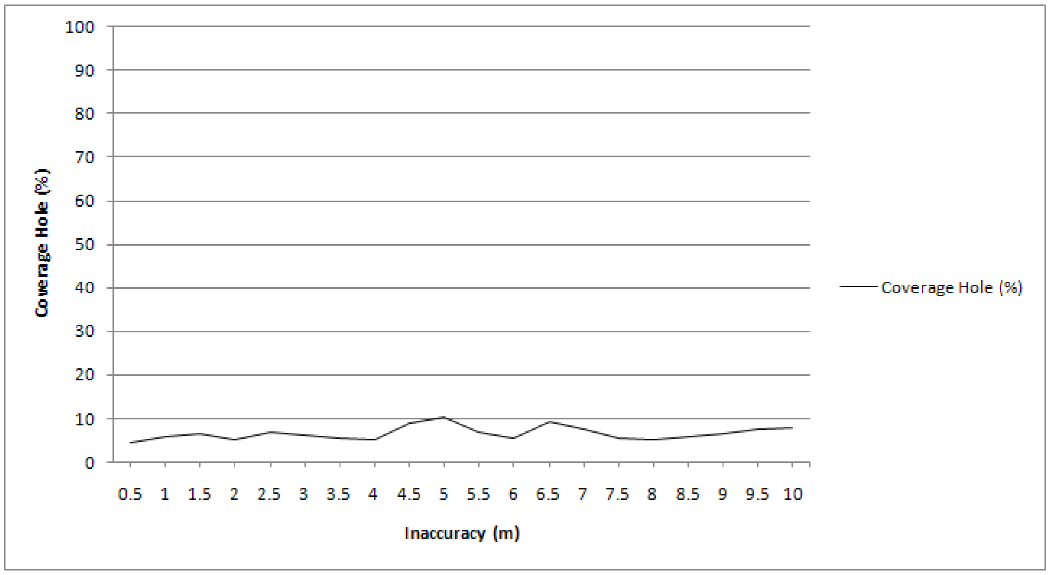


Figure 5-21 - Coverage hole(%) related to an increase in location inaccuracy (Bidding Protocol II).

Figure 5-21 shows the effects of complete inaccuracy injected into all the mobile nodes. The experiment is run with all the mobile nodes placed at degrees of inaccuracy varying between 0.5 and 10 meters. As the static nodes cannot be moved it would suggest that the worst case result of the algorithm would be to have the coverage hole remain at the initial value of 32%. The worst case shown in this experiment was a mere 10%, twice the value obtained in the control experiment, yet a third less than the worst possible case. For the result of 32% to be achieved all nodes would either need to be placed outside the ROI or completed overlapped by redundant nodes. This did not occur in the experiment, and would presumably only occur if random errors of much more than 10 meters were injected.

By design the algorithm results in a valid termination of the algorithm and a possible benefit in the case of location inaccuracy. When a mobile node is moved into position, to heal the coverage hole left by the static node, the mobile node is given the healed coverage hole as its new weight or bidding value. In implementing the algorithm in the presence of injected inaccuracies in the positioning of mobile nodes, it was necessary to make a design decision about the following matter. When a mobile node is moved into position to heal the coverage hole left by the static node, it is given the healed coverage hole as its new weight or bidding value. It was decided to set this value as if the mobile node had been accurately positioned. If this were not to be the case the algorithm could run infinitely, as coverage holes may never be resolved thereby always leaving the termination criterion unsatisfied. This infinite running could potentially see nodes move outside the ROI in much the same way as that of the VEC algorithm previously.

As with the previous variation, the termination criterion experiment is omitted from this algorithm.

5.2.3 Conclusion

A hybrid network attempts to reduce costs and complexity of the network by introducing a combination of mobile and static nodes. In this section the study compared the effects of location inaccuracies with respect to both the static nodes and the mobile nodes. When inaccuracies are introduced to the mobile nodes the algorithm appears to behave in an acceptable manner, meaning that the number of inaccurate mobile nodes directly relates to the effectiveness the nodes have on the network. Thus one can say that the network planner may have confidence in the network if majority of the mobile nodes are fairly accurate with all the static nodes being accurate. On the other hand if minor inaccuracies are placed within the static nodes, whether this is in the degree of inaccuracy or the number of inaccurate nodes, the algorithm appears to behave in a random fashion. The results of both experiments suggest that for the network to perform optimally, the static nodes need their position accurately reported within the ROI, while limited inaccuracies in the mobile nodes are tolerable.

5.3 Static Algorithms

The following section examines the results obtained for the two static algorithms. As the static algorithms do not move nodes but simply reduce the number of nodes sensing the network at a given time, the algorithm's assessment included the number of nodes active at that time in combination with the actual coverage of the ROI.

The control and initial deployment sensitivity experiments assessed the algorithm based on the number of deactivated nodes. The reason for not assessing the overall coverage hole in these experiments is that if the coverage hole were to fluctuate then this would suggest that the algorithm is not implemented correctly. The coverage hole is still checked in these experiments as way of ensuring the correct implementation of the algorithm. One would expect the coverage hole to remain the same as the algorithms goal is to maintain coverage with minimum node activation.

Subsequent experiments assessed the algorithms based on changes in the overall coverage hole. By examining the coverage holes the study assessed the sensitivity of the algorithms to node location inaccuracy.

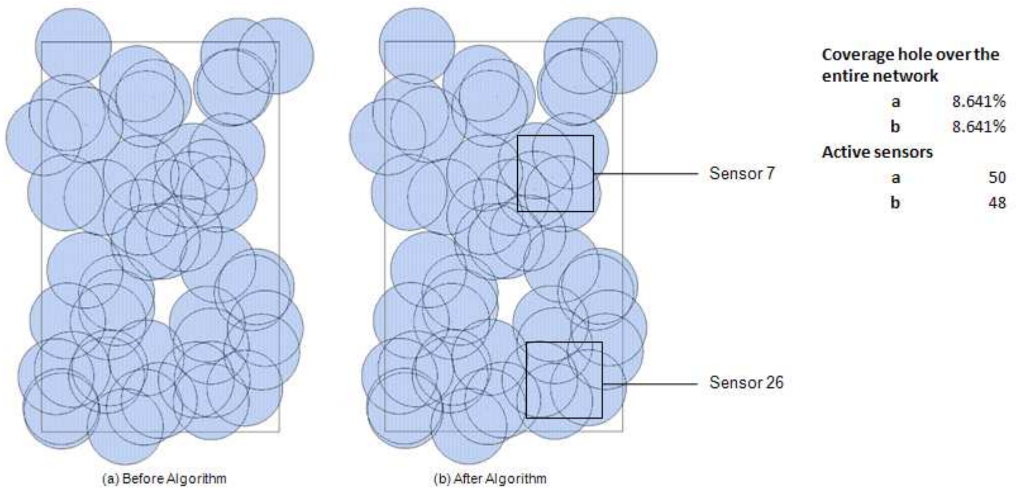
As discussed in Chapter 4, the number of nodes is doubled to 50 in the static algorithms. The reason for increasing the number of nodes is to cause overlapping of node's sensing areas. By introducing overlapping in the sensor area of nodes, redundancy is introduced into the network.

5.3.1 Coverage-Preserving Node Scheduling Scheme (CPNSS)

The CPNSS algorithm presented by Tian *et al.* 2002, is one of the simplest algorithms in terms of processing and calculation. The algorithm determines a set of neighbouring nodes that are within a given nodes sensing area. If the given node is completely covered by its neighbours then that node is deactivated. Interestingly enough the simplest algorithm has resulted in the most robust of all the algorithms implemented in this study - so much so that it appears that the algorithm is completely immune to location inaccuracies within the ranges that were examined.

Figure 5-22 shows a visual result of the algorithm's single iteration. The execution of the algorithm over a single iteration is a result of the following: since the locations are constant, the outcome of calculating the neighbours is always the same. As the simulator does not take into account energy consumption and remaining energy of nodes, nodes are deemed to be fit to support their neighbouring nodes 100% of the time. (Refer Section 4.2.3.1 to for details on the algorithm.)

The figure shows the algorithm powering down two nodes within the network (Sensors 7 and 26) whilst maintaining coverage of 91.36%. Since a two-node redundancy did not seem like a significant number of nodes to power down, a subsequent experiment containing a larger dataset was run later in this study. However, the present configuration was retained for experiments to test the algorithm's sensitivity to the initial positioning of nodes, and to node location inaccuracy.



Coverage hole over the entire network

a	8.641%
b	8.641%

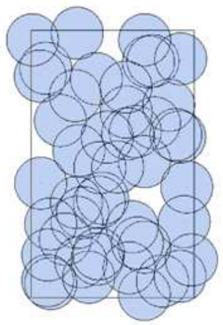
Active sensors

a	50
b	48

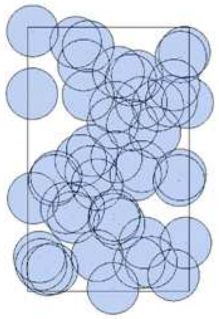
Figure 5-22 - Execution of the CPNS Algorithm for control purposes

Figure 5-23 shows the impact the algorithm had on a further fifteen network layouts. In each case, initial placement of nodes was randomly determined. The layouts range in degree of coverage hole percentages as well as in the extent of network clustering. (Network clustering is a reference to groups of nodes forming in the network with greater degrees of sensing area redundancy within the groups.)

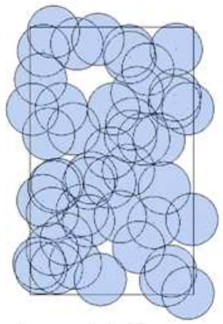
The algorithm powers down an average of 2 nodes, or 4% of the network across the various scenarios.



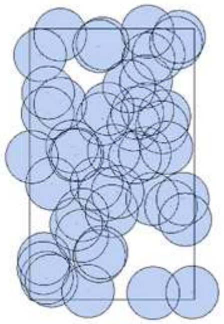
Coverage hole %	
Start	4.970%
End	4.970%
Active sensors	
Start	50
End	49



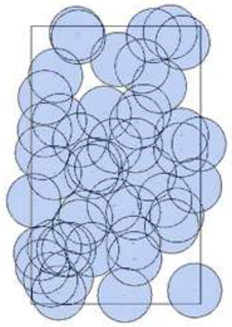
Coverage hole %	
Start	12.854%
End	12.854%
Active sensors	
Start	50
End	49



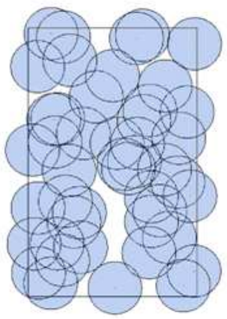
Coverage hole %	
Start	6.3179%
End	6.3179%
Active sensors	
Start	50
End	48



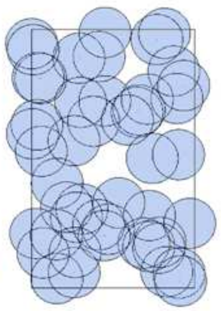
Coverage hole %	
Start	11.782%
End	11.782%
Active sensors	
Start	50
End	50



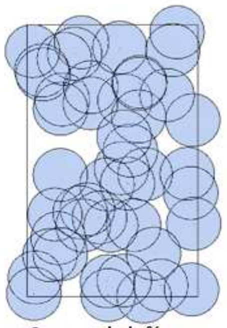
Coverage hole %	
Start	7.973%
End	7.973%
Active sensors	
Start	50
End	49



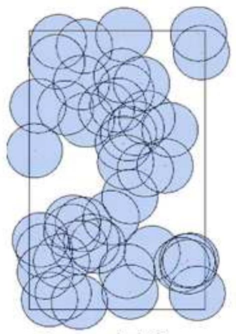
Coverage hole %	
Start	7.780%
End	7.780%
Active sensors	
Start	50
End	46



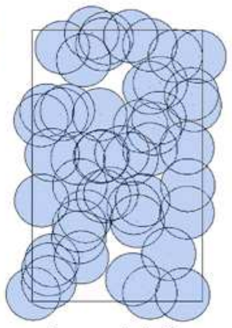
Coverage hole %	
Start	12.004%
End	12.004%
Active sensors	
Start	50
End	48



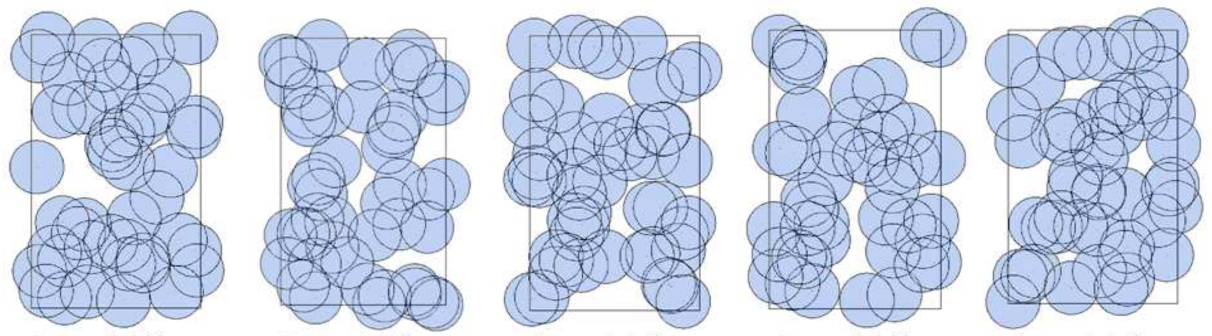
Coverage hole %	
Start	9.724%
End	9.724%
Active sensors	
Start	50
End	48



Coverage hole %	
Start	18.046%
End	18.046%
Active sensors	
Start	50
End	47



Coverage hole %	
Start	7.025%
End	7.025%
Active sensors	
Start	50
End	50



Coverage hole %					
Start	10.004%	11.585%	12.885%	17.330%	8.661%
End	10.004%	11.585%	12.885%	17.330%	8.661%
Active sensors					
Start	50	50	50	50	50
End	46	48	49	44	50

Figure 5-23 - Fifteen random starting position.

As with the previous algorithms, Figure 5-25 and Figure 5-26 introduce location as a factor. As with the previous algorithms, location inaccuracies in node positions at distances ranging from 0.5 metres to 10 metres (in increments of 0.5 metres) were systematically introduced. Again, the number of inaccurate nodes were varied from 0 to 25 nodes. The results of the experiment showed that the algorithm was entirely immune to location inaccuracies—i.e. within the range of the given inaccuracies, inaccurate locations did not impact the algorithm whatsoever.

This would appear to be due to one key factor: the algorithm uses the union of the areas of the neighbouring nodes to calculate whether a node should be powered down. If it is perceived that the node is not covered by its neighbours then it remains active. Figure 5-24 illustrates this in practice. Figure 5-24 (a) shows the sensors when the neighbours are accurately chosen. As can be seen sensor S_i is completely covered by its neighbouring nodes (shown as the grey area around the sensing area) and for this reason sensor S_i would be deactivated. Figure 5-24 (b) shows the same scenario with a single sensor, S_3, broadcasting its node inaccurately. The union of the neighbouring nodes is calculated as two separate geometric polygons with S_i not falling within the neighbour's bounds. The result is that S_i remains active. The same applies to nodes that falsely find themselves in S_i sensor area. For sensor S_i to deactivate it would take at least three nodes to be inaccurately placed within S_i's sensing area in such away that their sensing areas would cover S_i. This did not seem to occur in any of the experiments performed.

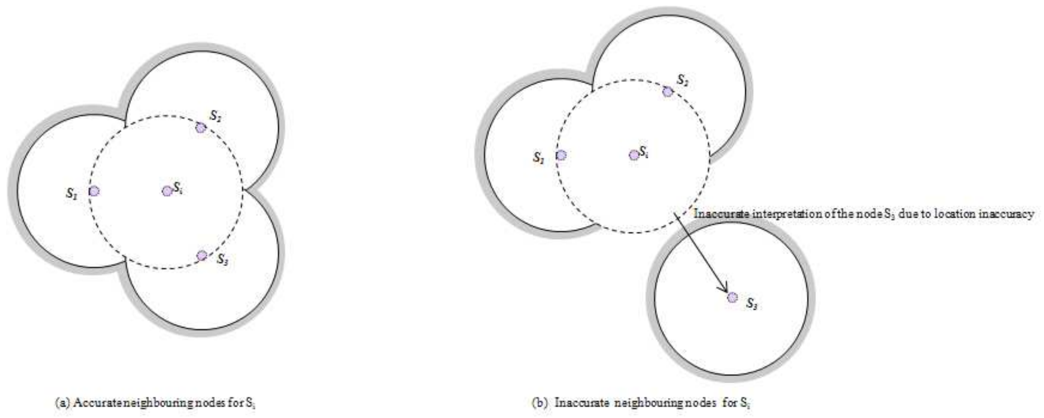

S_2
S_1
S_i
S_3
(a) Accurate neighbouring nodes for S_i
S_2
S_1
S_i
Inaccurate interpretation of the node S_3 due to location inaccuracy
S_3
(b) Inaccurate neighbouring nodes for S_i

Figure 5-24 - Location inaccuracy in determining node neighbours.

Even if the algorithm were to fail at deactivation of nodes, with no nodes powering down whatsoever, coverage of the original network would remain the same.

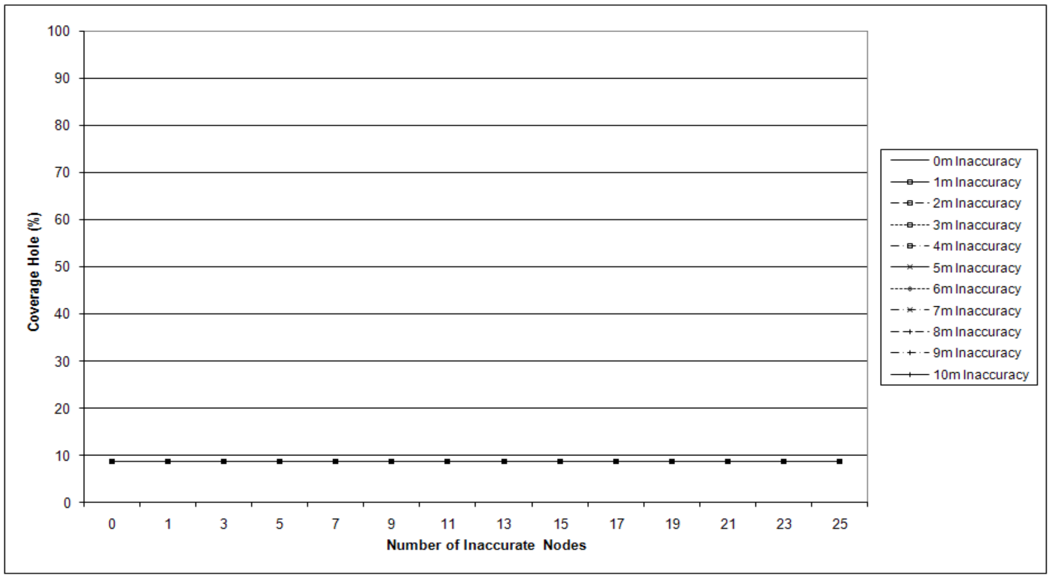


Figure 5-25 - Increasing the percentage of inaccurate nodes (CPNSS).

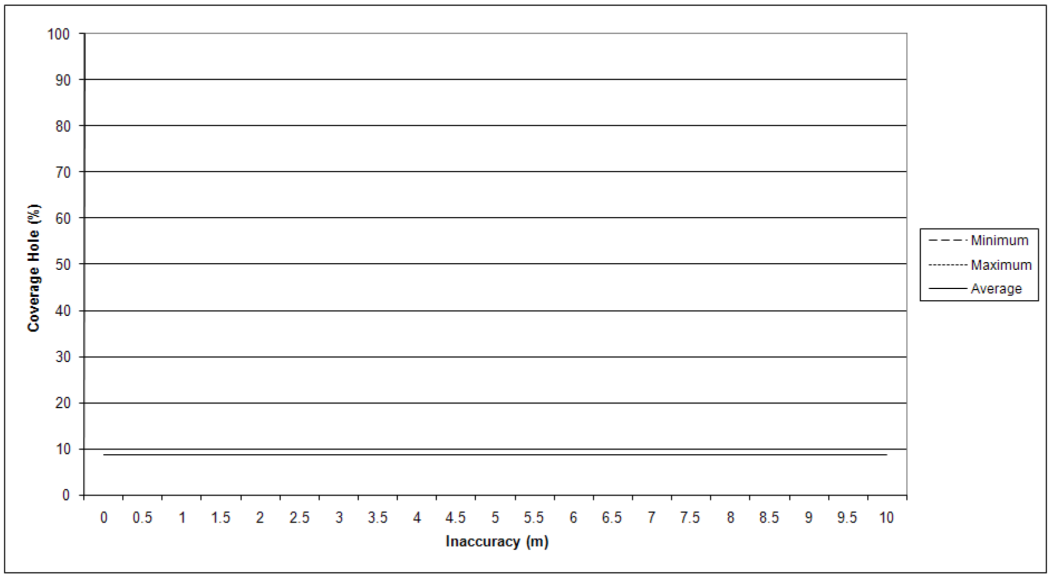


Figure 5-26 - Coverage hole (%) related to an increase in location inaccuracy (CPNSS).

Since the algorithm only involves one iteration as opposed to multiple iterations in the case of the hybrid and mobile algorithms, an analysis of the algorithm in terms of number of iterations does not apply. As this study does not take into account energy consumption as the basis for choosing the starting nodes, each iteration will choose the same nodes to power down.

As the algorithm would only be implemented as a single iteration, no termination criterion is put in place. For this reason a test for *Termination Criterion Sensitivity* did not apply.

As a final experiment, it was decided to run the algorithm with a larger network set of 100 nodes. The aim of the experiment was to assess whether or not a higher redundancy of nodes impacts the algorithm at all.

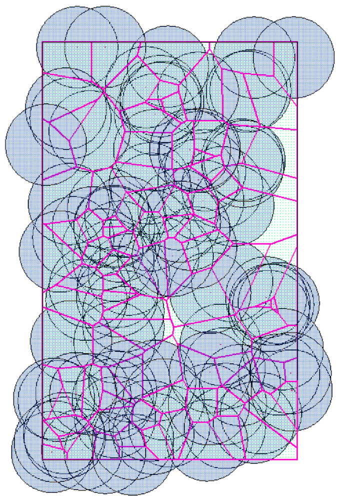


Figure 5-27 - Network with 100 randomly placed nodes.

Figure 5-27 shows the network with 100 randomly placed nodes. The initial coverage hole in this network layout was 4.6% of the ROI. The experiment was run a total of 100 times, each time allowing a different node to initiate the algorithm. The experiment was run a further 20 times, each time with a different degree of inaccuracy in the same way outlined in Section 4.1.4. The algorithm was then assessed on average coverage hole obtained by the 100 iterations at each run.

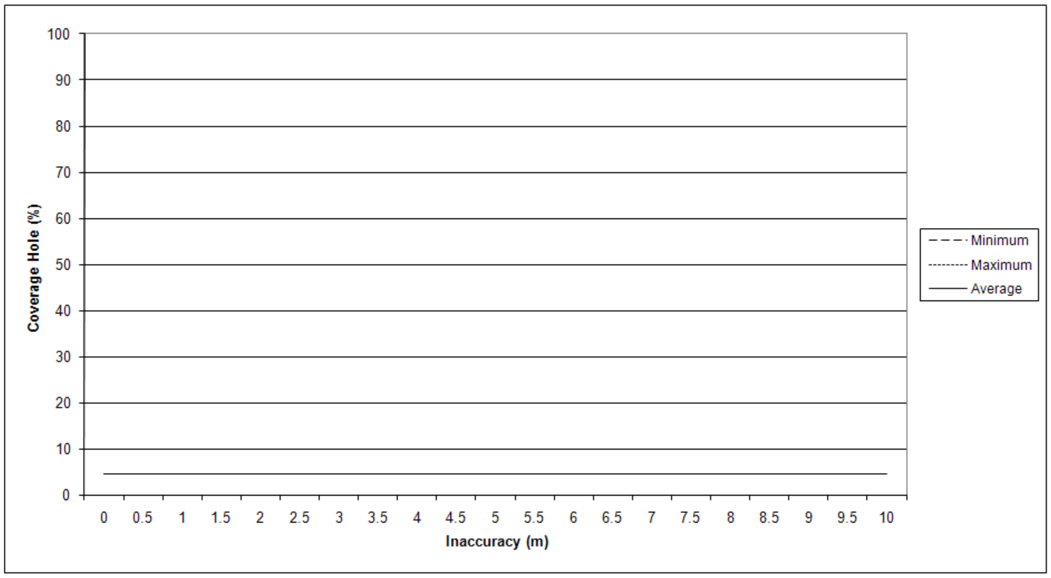


Figure 5-28 - Coverage hole (%) related to an increase in location inaccuracy (CPNSS) with 100 nodes.

As can be seen in Figure 5-28 there is no change in the experiments result when more nodes are introduced. Location inaccuracy appears to have no impact on the algorithm with the coverage hole remaining at 4.6% suggesting that no nodes are deactivated erroneously.

5.3.2 Optimal Geographical Density Control (OGDC)

The OGDC algorithm is slightly more complex to implement when compared to the CPNSS algorithm discussed above. CPNSS was limited, per definition of the algorithm, to a single iteration. The OGDC algorithm, in contrast, is started from a random position each time it is executed. For this reason more than one iteration can be executed. As opposed to starting the algorithm from a random node each time, as was done by the algorithm’s authors, in this study, it was decided to run the algorithm a total of 50 iterations, each time starting from a new node. By running the experiment in the following way, each node is allowed to start the algorithm, yielding an overall comparison of how the network would be represented if each node had an opportunity to start the algorithm. After each iteration the algorithm resets the network (nodes status of being ON or OFF). The control experiment’s starting scenario the same as that used in the CPNSS algorithm above.

Figure 5-29 shows the control experiment over the 50 iterations, with each iteration allowing a different node to be the starting node for the algorithm. (Note that, as expected, the coverage hole remained constant at 8.64%.) The information in the figure can be summarised by noting that the algorithm deactivated an average 0.62 nodes, that the minimum number of deactivated nodes was 0, and that the maximum number was 5. The OGDC's average number of deactivated nodes is thus slightly lower than that of CPNSS and the maximum number of deactivated nodes is somewhat higher than that of the CPNSS algorithm. The OGDC algorithm might therefore sometimes be more efficient in that it might deactivate more nodes than that of the CPNSS algorithm. However, optimal node deactivation takes place if and only if the correct node is chosen as the starting node, and this cannot be guaranteed.

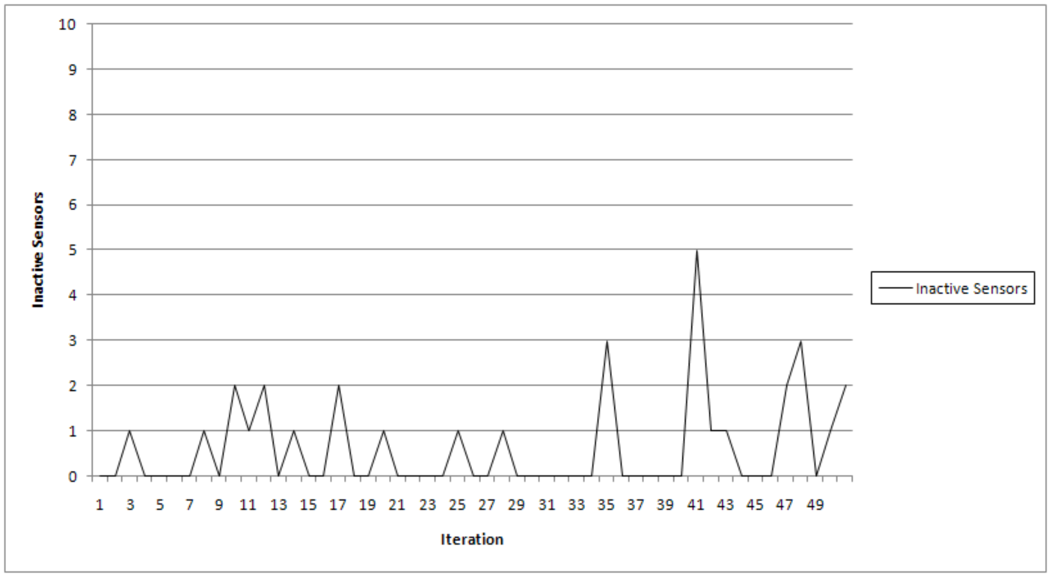


Figure 5-29 - Control experiment over 50 iterations.

As in the CPNSS algorithm, the OGDC was also executed over 15 other starting scenarios. As with the control experiment, 50 runs were executed, allowing all sensors to be used as the starting node. Figure 5-30 shows the minimum, maximum and average number of inactive nodes for each starting scenario.

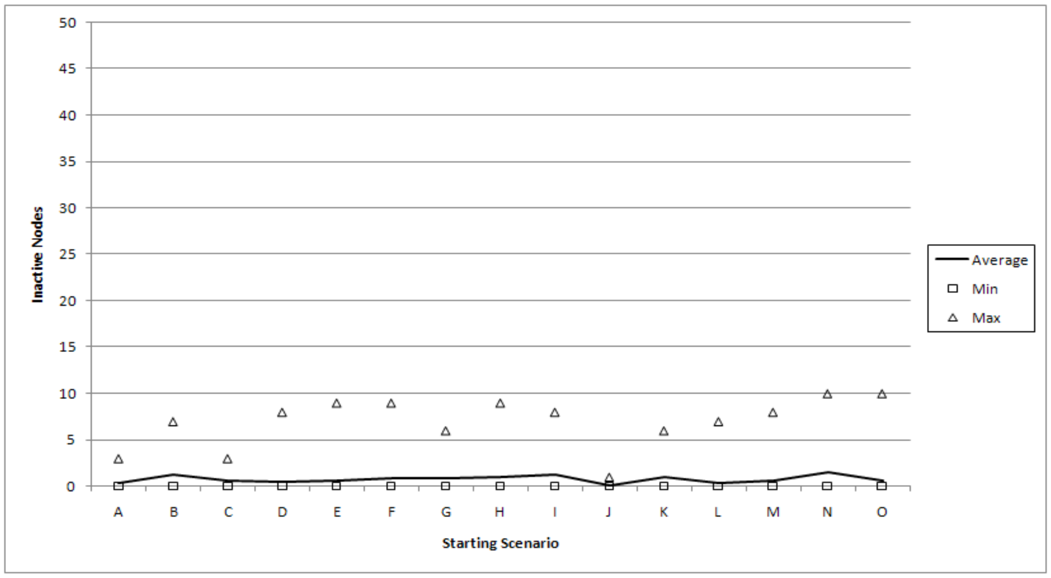


Figure 5-30 - Alternative randomly deployed starting scenarios for the OGDC algorithm.

The minimum value always stays at 0 as some starting scenarios may not yield any inactive nodes. The maximum value of inactive nodes averages around the 7 node mark. The maximum values obtained occurred on a very minimal basis, sometimes only once throughout the 50 iterations. The number of nodes to be deactivated in a scenario therefore appears to be rather sensitive to the starting node. In some scenarios, the maximum number of deactivated nodes could only be determined from one unique starting position—all other starting positions resulted in fewer deactivated nodes.

Interestingly though is that the average number of inactive nodes, approximately 2 is similar to that of the control experiment. The average maximum number of inactivate nodes is also similar to that of the control experiment at approximately 7 nodes. It can then be said that the various starting scenarios behave in a similar way to that of the scenario used in the control experiment.

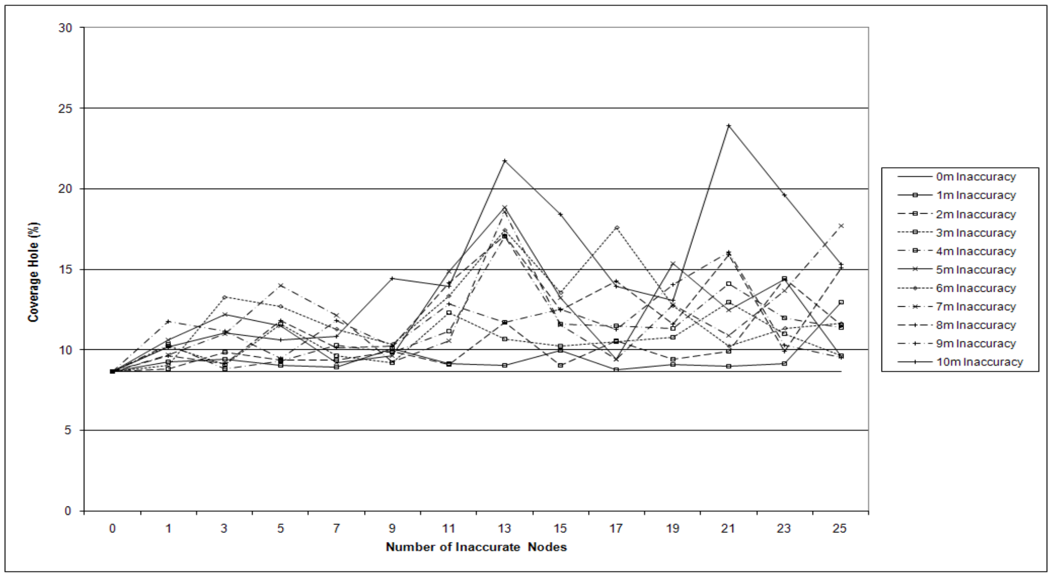


Figure 5-31 - Increasing the percentage of inaccurate nodes (OGDC).

Figure 5-31 shows the overall coverage hole within the control experiment's network when the nodes are exposed to various degrees of inaccuracy. Inaccuracy is introduced to a percentage of nodes starting with no nodes being inaccurate and ending with all 25 nodes inheriting a degree of location inaccuracy. It can be seen that as the number of inaccurate nodes increase so does the coverage hole percentage – as with the other algorithms implemented for mobile and hybrid network topologies. Interesting is the fact that with only 1 inaccurate node in the network, the network shows vulnerability to inaccuracy.

The final experiment attempts to assess the algorithm's performance when all nodes in the network are rendered increasingly inaccurate. Again, in each case the algorithm was run a total of 50 times, each time commencing with a different starting node. The graph shows the minimum, maximum and average coverage hole within the ROI. As the static algorithms do not increase the degree of coverage, it is not unexpected to see that the minimum coverage hole remains at 8.64%. Consistent with the previous experiment, the average coverage hole increases slightly with the increase of location inaccuracy. The maximum values are interesting to note. As the inaccuracy increases to 10m, the algorithm inappropriately deactivates nodes causing larger coverage holes, with a peak at 72.3%.

To explain these high peaks, note that each node's coverage represents approximately 4.5% of network coverage. This statement is based on a node with no overlapping coverage of a neighbouring node and complete coverage falling within the ROI. Thus, for each incorrect deactivate of a node, there is a potential loss of 4.5% coverage in the network.

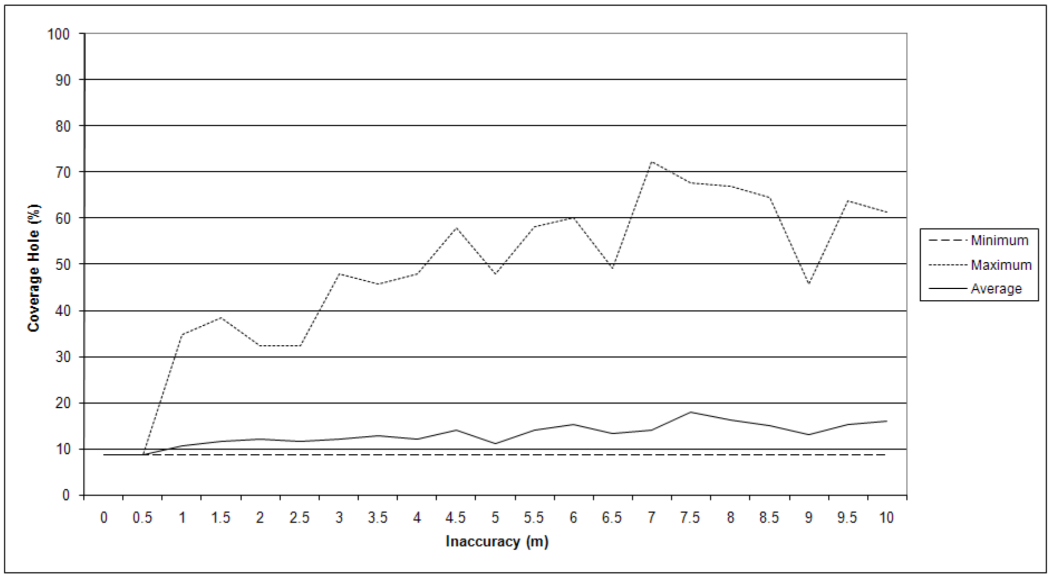


Figure 5-32 - Coverage hole (%) related to an increase in location inaccuracy (OGDC).

Since the notion of a termination criterion does not apply to this algorithm, the previous experiments relating to termination criteria are not relevant for this algorithm.

As with the CPNSS algorithm is was deemed appropriate to assess the algorithm with a larger network size. The same network layout used in Figure 5-27 was used to assess the OGDC algorithm.

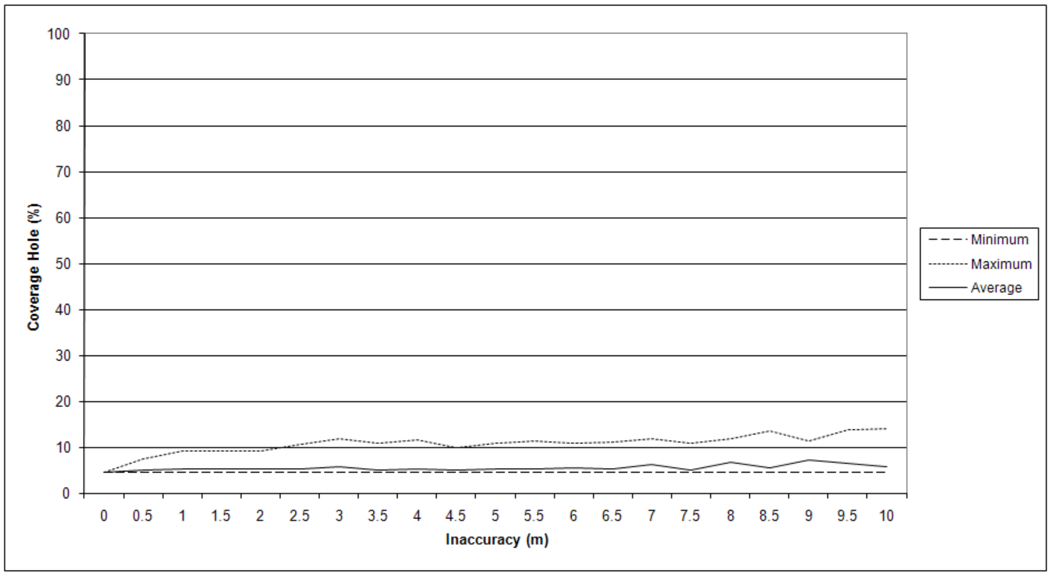


Figure 5-33- Coverage hole (%) related to an increase in location inaccuracy (OGDC) with 100 nodes.

As seen in Figure 5-33, the maximum coverage holes obtained at the various inaccuracy levels around 10%, in contrast with the spikes seen in the previous experiment. In this sense, the algorithm appears to be more stable with a larger network size. This is as a result of the higher degree of redundancy and overlapping of nodes within this network layout. In a scenario where a node is inappropriately deactivated, there exists a larger group of nodes that are still sensing that given area. If the study were then to assume that the worst degree of coverage hole obtained by the algorithm is an average of 10%, then one can see that the algorithm is still vulnerable at low degrees of inaccuracy. A maximum coverage hole of 6% is reached when the nodes are a mere 0.5m inaccurate. This same behaviour was seen in the previous experiment when a maximum coverage hole of 35% was reached at the same degree of inaccuracy. With the larger network set the algorithms average coverage hole percentage is very close to the starting coverage hole percentage of 4.6%.

5.3.3 Conclusion

The above assessment compared two static algorithms. Under accurate location information, static algorithms do not increase the network coverage within the ROI but simply attempt to reduce the number of active nodes. The OGDC algorithm was shown have the potential of reducing the number of active nodes in the network whilst maintaining the same degree of coverage to a better extent than that of the CPNSS algorithm. However, this advantage depended on the starting node for the algorithm and could not be guaranteed if an arbitrary starting node was selected.

OGDC was also found to depend more critically on the accuracy of location data. A single node deactivating in the network unintentionally results in a loss of coverage and an increase in the overall coverage hole percentage. The CPNSS algorithm was found to be immune to location inaccuracies within the limits of the various configurations tested: the algorithm's overall objective of deactivating nodes is accomplished without any loss of network coverage, irrespective of location inaccuracies. From this perspective the CPNSS algorithm is superior to the OGDC.

5.4 Chapter Conclusion

In this Chapter the study analysed the results obtained by the experiments outlined in Chapter 4. The chapter collected empirical information about algorithms classified into the three network deployment types: mobile, hybrid and static. It showed the key differences between algorithms within these network deployment types. The following chapter presents the concluded results obtained by these experiments. The chapter provides the answers to the questions proposed in Chapter 1.

Chapter 6

Conclusion

"No matter how many instances of white swans we may have observed, this does not justify the conclusion that all swans are white."

Sir K. R Popper – The Logic of Scientific Discovery 1934

6.1 Conclusion

Wireless Sensor and Actuator Networks (WSANs) are a key topic of interest in present day computing. In Chapter 1 it was noted that accurate location of nodes was a key assumption made in all WSAN node deployment algorithms. The study posed two questions: Firstly, what are the effects of location inaccuracies on the networks? Secondly, how do the various network deployment types compare? Before seeking answers to these questions, technical background was provided in Chapter 2 of this study, while Chapter 3 looked at the role that GIS simulation could play in the ongoing WSAN research.

The consequences of the accurate node location assumption became evident in the assessment of the algorithms, specifically node placement algorithms relating to mobile networks. Both the *VOR* and the *VEC* algorithms were affected by location inaccuracies to some degree, the *VOR algorithm* more so than the *VEC algorithm*. The study showed that at a low degree of inaccuracy the VOR algorithm behaved in an acceptable, somewhat robust manner. The VEC algorithm appeared to have a greater dependency on location accuracy. An advantage of the VEC algorithm over the VOR algorithm is that the VEC algorithm takes into account the boundary. By including the boundary the nodes are contained within the ROI at all times. In the case of VOR the nodes could potentially drift outside the ROI.

The study implemented a single algorithm to solve coverage within hybrid networks. This algorithm, *bidding protocol*, was then run under two forms of node location inaccuracy: from the viewpoint of the mobile node location inaccuracy and then the static node location inaccuracy. The algorithm appeared to be stable only when the static nodes were deemed to be accurately placed and the mobile nodes did not exhibit large degrees of inaccuracy. Both these requirements are fairly demanding, in the sense that in practice, static nodes may be inaccurately placed and likewise the mobile nodes may contain larger degrees of inaccurate placement. It could be said that the hybrid algorithms appeared to be more dependent on accurate node locations than the other algorithms that were investigated.

Finally the static network topology was assessed using two algorithms, namely the *CPNSS* and *OGDC* algorithms. Interestingly enough, the static algorithms appeared to be the most stable. Within the experimental limits of the study, the CPNSS algorithm was completely unaffected by the inaccuracies. The second algorithm, OGDC, performed well in deactivating nodes compared to CPNSS. Although this latter algorithm also exhibited a dependency on the node location accuracy, but this dependency did not seem to increase as the location inaccuracy did.

In general it is difficult to assess or compare the three network topologies. Due to the cost difference in networks as well as the landscape within which the topologies can be deployed, the use of the three kinds of topologies will differ in the real-world. Other factors aside, in terms of location dependencies, the static algorithms behaved in such a way that they would be recommended when deciding between network topologies.

The results of this study are useful in two ways. Firstly, they indicate the degree of confidence that the network planners may have in the deployment algorithms in the context of their dependency on accurate location information. Secondly, the results suggest an approach to node placement in the first place: simulate the optimal positioning of nodes as has been done in this exercise. Then attempt to place the nodes in locations suggested by the simulation, but with a degree of confidence that moderately erroneous placement, to the extent indicated by the study, is unlikely to have a great impact on the extent of coverage.

6.2 Related Work

As this study has shown, location data plays an important role in network deployment and reconfiguration. Work done by de Silva and Ghrist (2007) concentrates on Homological Sensor Networks (HSN). These are networks whose nodes, though in communication with their neighbours, are unaware of precise neighbour locations. Invariant theorems about homology groups (homology is characterised as measuring “certain types of chains, or objects built from simple oriented pieces”) indicate the presence of coverage holes within the ROI, without the need for location data. A current shortcoming in their solution is that nodes on the boundary have to be aware of their status as boundary nodes. Potential failure of nodes on the boundary would require a recalculation of the boundary nodes. The calculation of the boundary nodes is seen as an example of the convex hull problem as described in Harel (1996). This problem is considered to be polynomial, the complexity of the algorithm is then estimated to be *n*, where *n* is the number of nodes in the network.

6.3 Future Work

WSAN research is a new and emerging field. There currently exist many branches of research, varying from energy management and control to cost-effective construction. An area of unexplored research relates to extending the simulator to deal with the effects of real-world objects on a WSAN layout. A key direction would be to introduce radio attenuation and interference and to examine the effects that these have on optimal placement. Current research omits these real-world objects and assumes a uniform sensing area. This will not be the case in a ROI where buildings cause radio frequencies to be scattered or shadowed. Areas in the network may not be covered due to shadows, resulting in areas in the network that radio signals cannot reach. Another area of interest is to consider the effects that different mediums, for example soil and water, have on radio frequencies. For example, if sensors are used underground, or for deep-sea research, these different mediums may cause the radio signals to behave differently.

This study only assessed the major or more popular relocation algorithms. Further studies may include some of the other algorithms mentioned within this work.

With regards to hybrid networks, an algorithm is needed to determine the optimal ratio of static to mobile nodes during the deployment phase of a hybrid sensor network.

Furthermore, future studies should consider other mobile networks that use virtual forces or sequential approaches to calculate the placement of nodes. Virtual force algorithms compute forces to push the nodes away from each other, where each node acts as a virtual magnetic particle. All nodes, as well as the boundary, push neighbours away from each other in much the same way as the VEC algorithm presented earlier. The key difference is the force is not calculated as a result of the location of the neighbour but more of a force such as that exhibited by magnets. Nodes use the radio signals of their neighbouring nodes and strength thereof to determine the distance to each other. The idea behind this approach is that the nodes would self organise in a uniform position within the network.

The sequential approach, on the other hand, calculates the new locations of nodes based on the previous deployment of nodes within the networks. Each node is placed one-by-one in the network. Working in much the same way as the mobile nodes used in the bidding protocol, this approach incrementally deploys nodes, with each node's place based on information gathered from the current nodes within the network (Yang *et al.* 2006).

7 Glossary

Actuator	A node that is able to act / change the region of interest
Attribute	A descriptive character mapped to a component of the topology, either a cell as in the case of Raster images or a geometric shape as in the case of Vector images.
AOA	Angle of Arrival, the angle at which a radio signal arrives at the destination. Used in radiolocation.
Bandwidth	Rate of data transfer measured in bit/sec
CASE Tool	Computer Aided Software Engineering tool, assists in the creation of Smallworld objects for use in the VMDS and Smallworld environment
CARIS	Computer Aided Resource Information Systems is a company that produces GIS and hydrographical software
CCP	Coverage Configuration Protocol, solution to coverage problems, specifically energy conservation in static sensor networks
CGIS	Canada Geographic Information Systems, created in the 1950 to assist in regulatory procedures for land-use management and resource monitoring
Co-Fi	Coverage Fidelity algorithm, using computational geometry the solution takes into account energy consumption to provide a possible solution to the coverage hole problem for mobile networks
Convergence	The process of the coverage algorithm approaching a limited value on coverage hole percentage
Coverage hole	Area within the network that is not covered by at most k sensors where k is amount of nodes required by the application
Databus	Provides data communication between application
Delaunay Triangulation	Given a set of points *P*, Delaunay triangulation is the process of triangulating all the points within the set such that no point in P is inside the circum circle of any other triangle
Deployment	The process of scattering nodes within the region of interest
DSS	Distributed Self-spreading algorithm, inspired by the equilibrium of molecules, is a solution to the coverage hole problem in mobile sensor networks. The algorithm works at reconfiguring the location of sensors within the network topology
ESRI	Environmental Systems Research Institute. Consulting firm that specialises in land use analysis projects

GENS	General Electric Network Solutions
GEO-Location	Determining the location of an object via the aid of GPS satellite transmission
GIS	A Geographical Information Systems is a systems capable of managing geo-referenced data
GPS	Global Positioning System. An electronic device used to determine location, direction, speed and time via satellite communication
GRASS	Geographic Resources Analysis Support System, open source GIS tool supporting both vector and raster data
Hybrid network	A WSA network that contains a mix of movement-assisted nodes and static nodes
IDCA	Intelligent Deployment and Clustering Algorithm, much the same as DSS, this algorithm uses clustering to further reduce the energy consumption within the network
IQR	Interquartile range, is a measurement of statistical dispersion
Iteration	A single completed phase of a simulation, including detection of coverage holes and the solving of the hole
LCGSA	Laboratory for Computer Graphics and Spatial Analysis, a research and development school associated with Harvard Graduate School of Design
Magik	Object Oriented programming language developed by SmallWorld to use with the SmallWorld application
Mobile Network	A WSA network that contains only nodes that are able to move within the region of interest
Mobile Sensor	A node that is capable of movement, either via a third party or self movement
Movement-adjustment	The process of calculating potential coverage before a move is made to solve a given coverage hole. If the new coverage is "worse" than the current coverage the movement is reconfigured
Movement oscillations	The back and forth movement of a node between two points
Mote	See *node*
Node	An element within a WSAN, either a sensor or an actuator
OGDC	Optimal Graphical Density Control, provides a potential solution to the energy consumption of nodes in a densely populated region of the network (inverse of coverage holes)

Outlier	An outlier is defined as an observation that appears to be inconsistent with the other observations in the data set
Oscillation control	The prevention of movement oscillations by recording the angle of the previous move. If the new angle to move is opposite that of the previous move, movement is halted
Parcel	A piece of land that is commercial viable, i.e. can be sold or has market value
Projection	This is the transformation of a map from a sphere to a 2-dimensional plane surface.
Radiolocation	Process of finding the location of a point with the aid of radio waves
Raster data	A GIS data type. The object is divided into cells each cell contains data about the area
Reconfiguration	The process of reorganising nodes within a WSAN for optimal distribution
Run	A collection of iterations making up a single simulation
Scale	The mathematical relationship between distances on a map and the actual distance between objects on the Earth. This also applies the size of an object as represented on a map as well as the actual size on the Earth
Scalability	Property of a network or system to handle growth in work load or expansion in size
Sensor	A node that is able to sense the local environment
Simulation	Multiple iterations of the algorithm terminated by a preset condition, run over multiple times or runs
Smallworld	A GIS application suite developed by General Electric
Static Network	A WSA Network that contains only nodes that are not capable of movement i.e. nodes that are fixed to their current locations
Static Sensor	A sensor that is fixed to its current location
TIN	Triangulated Irregular Network
TOA	Time of Arrival. The time taken for a radio signal to reach the destination. Used in radiolocation.
Topology	The arrangement of nodes within the sensor network

Triangulation	The process of calculating the location of a point using the properties of triangles and the laws of sine
UAV	Unmanned aerial vehicle.
USA-CERL	U.S. Army Corp of Engineering Research Laboratory, developers of the open source application GRASS
Vector data	A GIS data type. Data is mapped to a geometric polygon, line or point
VMDS	Version Managed Data Store, SmallWorld developed database
Voronoi-diagram	Given a set of points *P*. This is the process of decomposing a given area or plain into sections based. The sections or polygons are determined via the proximity to neighbouring points within the set *P*.
Voronoi polygon	The sections that make up a Voronoi-diagram. Each section or polygon displays the property that the area within the polygon are closest to the point making the polygon within the diagram
WSAN	Wireless Sensor and Actuator Networks are much the same as mobile *ad-hoc* networks, however it is able to sense and act within its environment.

1 References

[1] N. Ahmed, S.S. Kanhere, and S. Jha, "The Holes Problem in Wireless Sensor Networks: A Survey", *Mobile Computing and Communication Review*, NICTA, Sydney, Australia, 2005, volume 1, number 2.

[2] J. Albowicz, A. Chen and L. Zhang, "Recursive position estimation in sensor networks", *The 9th International Conference on Network Protocols,* November 2001, pg 35-41

[3] I.F. Alkyildiz, and I.H. Kasimoglu, "Wireless Sensor and Actor networks: research challenges", *Georgia Institute of Technology,* Georgia, USA, 2004.

[4] I.F. Alkyildiz, W. Su and Y. Sankarasubramaniam, "Wireless Sensor Networks: A Survey", *Computer Networks*, vol 38, issue 4, March 2002, pg 393-422

[5] F. Aurenhammer, "Voronoi Diagrams – A survey of a fundamental geometric data structure", *ACM Computing surveys,* 1991

[6] V. Barnett, T. Lewis, *Outliers in Statistical Data,* John Wiley and Sons, 1994 3rd edition.

[7] M.A. Batalin and G.S. Sukhtame, "Coverage, exploration and deployment by a mobile robot and communication network". *Telecommunication Systems Journal, Special Issue on Wireless Sensor Networks,* 26(2):181-196, 2004.

[8] M. Becker, Simulation Tool Comparison Matrix, November 2007

[9] K.Q. Brown, "Voronoi Diagrams from convex hulls", 1979, pg 223-228

[10] R.F. Chavez, "Generating and Reintegrating Geospatial Data", *ACM, June 2000*

[11] CompuWorld, "BP Pioneers Large-scale Use of Wireless Sensor Networks", 14 March 2005, http://www.computerworld.com

[12] J. Cortes, S. Martinez, T.Karatas and F. Bullo, "Coverage Control for Mobile Sensing Networks". *In Proceedings of IEEE Conference on Robotics and Automation (ICRA), May*, 2002.

[13] FOSS4G2008 Homepage - www.foss4g2008.org/, 2008

[14] C.A. Frazee, "World History, Volume One: The Easy Way", Barrons Educational Series, 1997, pg 12, ISBN 0812097653

[15] S. Ganeriwal, A. Kansal and M.B. Srivastava, "Self Aware Actuation for Fault Repair in Sensor Networks". *In Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, May 2004.

[16] GENS SmallWorld Core Documentation Version 4, GE Power Systems, 2003

[17] GIS Lounge Library Homepage, http://gislounge.com/, 2008

[18] R.H. Gűting, "An Introduction to Spatial Database Systems", *Spatial Database Systems of the VLDB Journal*, vol 3, no 4, October 1994

[19] N. Heo and P.K. Varshney, "An Intelligent Deployment and Clustering Algorithm for a Distributed Mobile Sensor Network". In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics,* volume 5, pages 4576-4581, October 2003.

[20] C.F. Huang and Y.C. Tseng, "The Coverage Problem in Wireless Sensor Networks" *In Proceedings of the 2nd ACM WSNA'03, September 2003.*

[21] W.E. Huxhold, "An Introduction to Urban Geographical Systems", *University of Wisconsin, Milwaukee, Oxford University Press*, 1991

[22] ICOMOS, "Robben Island", International Council on Monuments and Sites, Recommendation for Robben Island to be a world site, ICOMOS, September 1999.

[23] J. Jiang and W. Dou, "A Coverage-preserving density control algorithm for wireless sensor networks", *In ADHOC-NOW'04*, pg 42-55, July 2004

[24] K.M. Johnston, “Geo-processing and Geographic Information System Hardware and Software: Looking towards the 1990s”, *Geographical Information Systems for Urban and Regional Planning*, pg215-227

[25] M. Kennedy, “The Global Positioning System and GIS: An Introduction”, *Ann Arbor Press, Inc,* Michigan USA, 1996, ISBN 1-57504-017-4

[26] J.M. Khan, R.H. Katz and K.S. Pister, “Next Century Challenges: Mobile Networking for Smart Dust”, *In Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, pg 271-278

[27] X.L. Li, P.J. Wan and O.Frieder, “Coverage in Wireless Ad Hoc Sensor Networks”, *In IEEE Transactions on Computers*, vol 52, issue 6, June 2003, pg 753-763

[28] (a) S. Meguerdichian, F. Koushanfar, M. Potkonjak and M.B. Srivastava, “Coverage problems in wireless ad-hoc sensor networks”, *INFOCOM 2001, 20th annual joint conference of the IEEE Computer and Communications Societies, Proceedings IEEE*, Vol 3 April 2001, pg 1380 – 1387

[29] (b) S. Meguerdichian, S. Slijepcevic, V. Karayan and M. Potkonjak, “Localized algorithms in wireless ad-hoc networks: location discovery and sensor exposure”, *In Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, 2001, pg 106-116

[30] MicroStrain Homepage, http://www.microstrain.com/, 2007

[31] D. Niculescu and B. Nath, “Ad hoc positioning system (APS) using AoA”, *In proceedings of the IEEE INFOCOM, 2003*.

[32] (a) H.A.B.F. Oliveira, E.F. Nakamura, A.A.F. Loureiro and A. Boukerche, “Error Analysis of Localization Systems for Sensor Networks”, *Proceedings of the 13th annual ACM international workshop on Geographic Information Systems,* 2005, p71-78

[33] (b) H.A.B.F. Oliveira, E.F. Nakamura, A.A.F. Loureiro and A. Boukerche, “Direct Position Estimation: A Recursive localization approach for wireless sensor network”, *the 14th IEEE International Conference on Computer Communications and Networks,* San Diego, California, USA, Oct 2005, pp 557-562

[34] D. Harel, “Algorithmics – The Spirit of Computing”, 2nd edition, Addison-Wesley Publishing Company, 1996

[35] M. Perry, F. Hakimpour and A. Sheth “Analyzing Theme, Space, and Time: An Ontology-based Approach” *ACM International Symposium on Geographic Information Systems*, 2006, pp. 147-154.

[36] D. Righton, C.Mills, “Application of GIS to investigate the use of space in coral reef fish: comparison of territorial behaviour in two Red Sea butterfly fishes”, *International Journal of Geographical Infomration Science*, vol 20, Issue 2, Februaru 2006, pg 215 - 232

[37] A.G. Ruzzelli, M.J. O’Grady, G.M.P. O’Hare and R. Tyan, “Adaptive scheduling in Wireless Sensor Networks”, *In proceeding of WAC2005, the 2nd IFIP Workshop on Automatic Communication*, 2005

[38] P. Santi “Topology Control in Wireless Ad Hoc and Sensor Networks” *ACM Computing Surveys, Vol 37, No 2 ’05,* June 2005, pp. 164-194.

[39] G. Simon, M. Maroti and A. Ledeczi, “Sensor Network-Based Counter-sniper System”, *ACM Sensys Conference,* November 2004

[40] M. Sharifzadeh and C.Shahabi, “Supporting spatial aggregation in sensor network database”, *In proceedings of the 12th Annual ACM international Workshop on Geographic Information Systems*, Washington DC, November 2004, pg166-175

[41] SunSpots Homepage - http://www.sunspotworld.com/, 2007

[42] D. Tian and N.D. Georganas, “A Coverage Preserving Node Scheduling Scheme for Large Wireless Sensor Networks”. *In Proceedings of the first ACM WSNA* ’02, September 2002.

[43] P. Vinten-Johansen, “Cholera, Chloroform, and the Science of Medicine: A Life of John Snow”, Oxford University Press, US, 2003, ISBN 019513544X

[44] M. Wachowicz, “Object-Oriented Design for Temporal GIS”, Taylor & Francis, 1999, ISBN 0-7484-0831-2

[45] (a) G. Wang, G. Cao and T.L. Porta "A Bidding Protocol for Deploying Mobile Sensors". *In 11th IEEE International Conference on Network Protocol* ICNP'03, pages 315-324, November 2003.

[46] G. Wang, G. Coa and T.L. Porta "Movement-assisted Sensor Deployment" *In IEEE INFOCOM 2004, June 2004*.

[47] (b) X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless and C. Gill "Integrated coverage and connectivity configuration in wireless sensor networks". In *Proceedings of the ACM SenSys '03*, pages 28-39, Nov 2003

[48] E.W. Weisstein, "Triangle." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Triangle.html

[49] Xbow Homepage - http://www.xbow.com, 2007

[50] S. Yang, J. Wu and F. Dai, "Localized Movement-Assisted Sensor Deployment in Wireless Sensor Networks", *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference*, Oct 2006, pg 753-758.

[51] C. Yearsley, M.F. Worboys, P. Story, D.P.W. Jayawardena and P. Bofakos, "Computational support for spatial information handling: models and algorithms", *Innovations in GIS, The First National Conference on GIS Research*, UK, vol 1, pg 75-88

[52] M. Zhang, X. Du and K. Nygard, "Improving coverage Performance in sensor networks by using mobile sensors", *Military Communications Conference MILCOM 2005, IEEE*, Oct 2005, vol 5, pg 3335-3341

[53] H. Zhang and J.C. Hou, "Maintaining Sensing Coverage and Connectivity in Large Sensor Networks", *Technical Report UIUDCS-R-2003-2351*, Department of Computer Science, University of Illinois at Urbana Champaign, 2003.

A Addendum – Tables

A.1 Simulator Comparison Matrix (Becker 2007)

	TOSSIM	TOSSIM	ns-2	ns-2	GloMoSim (Qualnet)	OMNeT++ (Omnest)	OMNeT++	OMNeT++	OMNeT++	OMNeT++
Version	1.1.15cvs	1.1.15cvs	2.30		2.0 (3.9.5)	3.4b2				
+Tool Extension (Tool Version)		tython (1.1.15cvs)	(BonnMotion, BonnTraffic)	SensorSim	(BonnMotion)		Sensim 3.0	Mobility Framework 2.0pre3	NesCT 30-09-06	EWSNSIM 171105
Simulator/Emulator	Simulator, Emulator(AVR, MSP)	same	Simulator	Simulator	Simulator	Simulator	Simulator	Simulator	Emulator (TinyOS)	Simulator
Transferability of Code	Yes	same	No	No	No	No	No	No	Yes	Yes
License, Cost	GPL	same	GPL	GloMoSim: Academic, QualNet: Commercial	Academic Public License, Omnest: Commercial	Academic Public License	Academic Public License	Academic Public License	Academic Public License	Commercial, University Program
Architecture	Component-based	same	Object-oriented			Component-based	Component-based	Component-based	Component-based	Component-based
Platform	Windows, Linux	same	Windows, Linux, Sun, Mac		Windows, Linux, Sun, Mac	Windows, Linux, FreeBSD, Mac OS X	Windows, Linux, FreeBSD, Mac OS X	Windows, Linux, FreeBSD, Mac OS X	Windows, Linux, FreeBSD, Mac OS X	Windows, Linux, FreeBSD, Mac OS X
WSN Platforms supported	MSP, AVR	same								
Status	Supported	same	Supported	Not supported	GlomoSim: Not supported, Qualnet: Supported	Supported	Not supported	Supported, Free Community Support, Commercial Support (OmNest)	Supported	

	TOSSIM	TOSSIM	ns-2	ns-2	GloMoSim (Qualnet)	OMNeT++ (Omnest)	OMNeT++	OMNeT++	OMNeT++	OMNeT++
Programming language of the tool	Java	same	C++, OTcl		Parsec (C, C++)	C++, NED	C++, NED	C++		
Programming language of the models	nesC	same	C++, OTcl			C++, NED	C++, NED	C++, NED	nesC	nesC, C++, NED
GUI, API etc.	GUI: TinyVIZ/SimDriver	same	GUI: nam		Yes	GUI: TkEnv	GUI: TkEnv	gned, GUI: TkEnv	GUI: TkEnv	GUI: TkEnv
Scenario description format		Python script	OTcl			NED	NED	NED		NED
Parallel execution			Pdns	Pdns	Parsec	Yes: Parsim	Yes: Parsim	Yes: Parsim	Yes: Parsim	Yes: Parsim
Radio propagation models	Empirical, Fixed Radius		Included: Free space model, Two-ray ground model, Shadowing model External: realistic channel propagation by Wu Xiuchao, ricean propagation model by Ratish J. Punnoose		Two Ray, FreeSpace		Plain pathloss model	Plain pathloss model, : Free Space, NoLoss, Gilbert-Elliot		
Physical layer and antenna models	Empirical, Fixed Radius	+PacketLossRatios			SNR bounded, BER based with BPSK/QPSK modulation					Ideal unit gain antennas, PSK, 16-QAM, 256-QAM antenna models ANSim Tool Traces

	TOSSIM	TOSSIM	ns-2	ns-2	GloMoSim (Qualnet)	OMNeT++ (Omnest)	OMNeT++	OMNeT++	OMNeT++	OMNeT++
Mobility models	No Mobility	Scriptable Pathes	(BonnMotion: Random Waypoint, Gauss-Markov, Manhattan Grid, Reference Point Group)		RWP, Random Drunken, Trace based	No			Yes (CircleMobility, ConstSpeedMobility, LinearMobility, LineSegmentsMobilityBase, MassMobility, RectangleMobility)	Bonn Motion data, LOGO scripts (TurtleMobility), CircularMobility, LinearMobility, MassMobility, RectangleMobility, ConstSpeedMobility
Standards supported	802.15.4		AODV, OLSR, DYMO, 802.11, Bluetooth, Mobile IP	802.15.4, 802.11 (BonnTraffic: several Traffic models)	CSMA, IEEE 802.11, MACA, IP with AODV, Bellman-Ford, DSR, Fisheye, LAR scheme 1, ODMRP, WRP, TCP, UDP, CBR, FTP, HTTP, Telnet + more	802.11		802.11, Directed Diffusion with GEAR		
Supports Energy Consumption Research	with extension PowerTOSSIM	with extension PowerTOSSIM								
Statistical support (RNG, Batch Means, Confidence interval, LRE etc.)		same	(ns2measure)			Akaroa RNG, seedtool			exhaustive	exhaustive

	OPNET	Avrora	ATEMU	EmStar/EmTOS	SENS	J-Sim(JavaSim)	ModelNet/NISTNet	NESLSim	WiSeNet	JiST / SWANS	SwarmNet/Shawn	AlgoSenSim	Netwiser
Version	11.0	Beta 1.6.0	0.4	2.5	jan31-2005b	1.3 + patch4	0.99/2.0.12	N/A	0.51	1.0.6	Daily builds	0.9.2.2	0.2.5
+Tool Extension (Tool Version)	Wireless Module												
Simulator/Emulator	Simulator	Simulator/ Emulator(AVR)									Simulator		
Transferability of Code	No	Yes									No		
License, Cost	GPL						GPL	N/A		Academic	BSD		Commercial license or GPL
Architecture	Object-oriented	Emulate hardware directly									Object-oriented		
Platform		JVM									Windows, Linux, Any with Standard C++		
WSN Platforms supported		AVR											
Status	Supported	Supported but not being actively developed		Inactive (Last update 2005)							Supported	Pre-alpha	Supported
Programming language of the tool	C, C++	Java				Java				Java	C++		
Programming language of the models	C, C++										C++		
GUI, API etc.	GUI	No GUI									No GUI		Eclipse Plugin
Scenario description format	GUI	Plain Text											

	OPNET	Avrora	ATEMU	EmStar/EmTOS	SENS	J-Sim(JavaSim)	ModelNet/NISTNet	NESLSim	WiSeNet	JiST / SWANS	SwarmNet/Shawn	AlgoSenSim	Netwiser
Parallel execution	Yes										No		
Radio propagation models	Yes	Perfect, Fixed Radius									Fixed Radius, Radio Irregularity Model (RIM), Permalink		
Physical layer and antenna models		Ideal											
Mobility models	Yes	No									ns-2 mobility files		
Standards supported	802.11	802.11, 802.16, MANET, MobileIP	TinyOS MAC, any application										
Supports Energy Consumption Research	Three states Radio Model (Sleep, Rx, Tx)												
Statistical support (RNG, Batch Means, Confidence interval, LRE etc.)	exhaustive	BSD RNG, Batch Means	No										

	DiSenS	SENSE	ModelNet/NISTNet	NESLSim	WiSeNet	JiST / SWANS	SwarmNet/Shawn	AlgoSenSim	Netwiser	DiSenS	SENSE	COOJA	JProwler
Version		2.0	0.99/2.0.12	N/A	0.51	1.0.6	Daily builds	0.9.2.2	0.2.5		2.0	No release yet.	
+Tool Extension (Tool Version)													
Simulator/Emulator							Simulator					Simulator, Emulator (MSP)	
Transferability of Code							No					Yes	
License, Cost			GPL	N/A		Academic	BSD		Commercial license or GPL			Apache	
Architecture							Object-oriented					Cross-Level simulation. (Nodes may either be simulated or emulated)	
Platform							Windows, Linux, Any with Standard C++					Linux, Windows	
WSN Platforms supported													
Status							Supported	Pre-alpha	Supported			Actively developed	
Programming language of the tool						Java	C++					Java	
Programming language of the models							C++					C (or Java)	
GUI, API etc.							No GUI		Eclipse Plugin			GUI	

	DiSenS	SENSE	ModelNet/NISTNet	NESLSim	WiSeNet	JiST / SWANS	SwarmNet/Shawn	AlgoSenSim	Netwiser	DiSenS	SENSE	COOJA	JProwler
Scenario description format													
Parallel execution							No					No	
Radio propagation models							Fixed Radius, Radio Irregularity Model (RIM), Permalink					Fixed Radius, Ray-tracing multipath model	
Physical layer and antenna models												Ideal	
Mobility models							ns-2 mobility files					Supported, no models	
Standards supported												Glue driver abstraction	
Supports Energy Consumption Research													
Statistical support (RNG, Batch Means, Confidence interval, LRE etc.)													

	Viptos/VisualSense/PtolemyII	Sunflower	WISENSE	SenSor	GTSNetS	SensorSim	SimulAVR	Sidh	SWAN	TOSSF
Version	1.0			CVS	1.0				1.0.1 alpha	
+Tool Extension (Tool Version)										
Simulator/Emulator				Simulator	Simulator					
Transferability of Code				No	No					
License, Cost					GPL					
Architecture					Object -oriented					
Platform				Python VM	Linux, Windows					
WSN Platforms supported							AVR			
Status	Inactive (Last update 2005)				Supported	Withdrawn	Inactive (Last update 2004)	Inactive. (Paper in 2005.)		
Programming language of the tool			SDL	Python	C++					
Programming language of the models			SDL	Python	C++					
GUI, API etc.				GUI	Qt based animation tool					
Scenario description format					C++					

	Viptos/VisualSense/PtolemyII	Sunflower	WISENSE	SenSor	GTSNetS	SensorSim	SimulAVR	Sidh	SWAN	TOSSF
Parallel execution					Yes, libsync library					
Radio propagation models					Unit Disk Graph, Two-ray ground model, Free space					
Physical layer and antenna models					ideal					
Mobility models					Random-waypoint and variants					
Standards supported					IEEE 802.3, IEEE 802.11, DSR, TCP, UDP, CBR, FTP, HTTP +more					
Supports Energy Consumption Research										
Statistical support (RNG, Batch Means, Confidence interval, LRE etc.)					Contains models for a variety of random number generators, including exponential, pareto, uniform, normal, empirical, constant, and sequential. Supports data collection using histograms and cumulative distribution functions.					

B Addendum - Development API

B.1 Simulator

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.init()

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.commit()

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.reset_roi(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.clean_roi(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.random_deploy(p_config, _optional p_sensor_amount, p_name, p_rx_seed, p_ry_seed)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.deploy_sensors_for(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.create_voronoi_diagram(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.clean_sensors(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.get_optimal_sensor_count(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.get_stats_for(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.set_mobility(p_config, p_mobile)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.get_total_dist(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.get_mean_dist(p_config)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.get_protocols_from(p_filename)

_pragma(classify_level=basic, topic={simulator,app,engine})
_method **simulator_engine**.store_data(p_config)

```
_pragma(classify_level=basic, topic={simulator,app,engine})
_method simulator_engine.export_data( p_config, p_name )

_pragma(classify_level=basic, topic={simulator,app,plugins})
simulator_manager.def_property ( :engine_name, :default, :simulator_engine )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.init ( _optional p_plugin, _gather p_args )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.build_gui ( p_container )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.init_actions()

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.manage_actions()

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.handle_actions( p_method, _gather p_args)

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.refresh_stats( p_config )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.set_mobility( p_config )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.random_deploy( p_config )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.run_protocol( p_config )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.export_data( p_config )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.toggle_run_inaccurate( p_config )

_pragma(classify_level=basic, topic={simulator,app,plugins})
_method simulator_manager.help_wanted( _optional p_id )

_pragma(classify_level=basic, topic={simulator,geometry})
_method common_math_mixin.get_xy_for_triangle( p_distance, p_angle )

_pragma(classify_level=basic, topic={simulator,geometry})
_method common_math_mixin.get_xy_for_boundary( p_distance, p_angle )

_pragma(classify_level=basic, topic={simulator,geometry})
```

_method **common_math_mixin**.get_xy_from_trail(p_distance, p_coord_a, p_coord_b)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.new()

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.construct_empty_circles(p_config)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.find_neighbour_for(p_sensor, p_set)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.find_vp_neighbours_for(p_tin, p_sensor)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.find_direct_neighbours_for(p_sensor, p_set)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.find_circumcenter_for(p_set)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.construct_diagram(p_config)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.get_diagram_coverage_hole(p_config)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.calculate_uniformity(p_config)

_pragma(classify_level=basic, topic={simulator,geometry})
_method **voronoi_diagram**.calculate_local_uniformity(p_config, p_si)

B.2 Protocols

_pragma(classify_level=basic, topic={simulator,protocol})
coverage_protocol.define_shared_variable(:name, "", :public)

_pragma(classify_level=basic, topic={simulator,protocol})
_abstract _method **coverage_protocol**.init(p_config)

_pragma(classify_level=basic, topic={simulator,protocol})
_abstract _method **coverage_protocol**.run()

_pragma(classify_level=basic, topic={simulator,protocol})
_abstract _method **coverage_protocol**.reset()

_pragma(classify_level=basic, topic={simulator,protocol})
_abstract _method **coverage_protocol**.stop()

_pragma(classify_level=basic, topic={simulator,vec,protocol}) _method **coverage_protocol**.move_sensor(p_config, p_sensor, p_coord)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **coverage_protocol**.get_coverage_for?(p_sensor)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.init(p_config)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.run()

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.discovery(p_config)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.get_neighbours_for(p_config, p_sensor, p_tin, p_ht)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.calculate_boundary_force_for(p_roi)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.moving(p_nsi, p_si, p_davg)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.calculate_average_for(p_config)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.reset()

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vec_protocol**.stop()

_pragma(classify_level=basic, topic={simulator,vor,protocol})
_method **vor_protocol**.init(p_config)

_pragma(classify_level=basic, topic={simulator,vor,protocol})
_method **vor_protocol**.run()

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vor_protocol**.get_farthest_vertex(p_si)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vor_protocol**.move_sensor(p_si, p_d_asi, p_sl, p_v_coord)

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vor_protocol**.moving()

_pragma(classify_level=basic, topic={simulator,vec,protocol})
_method **vor_protocol**.discovery(p_config)

_pragma(classify_level=basic, topic={simulator,vor,protocol})
_method **vor_protocol**.reset()

_pragma(classify_level=basic, topic={simulator,vor,protocol})
_method **vor_protocol**.stop()

_pragma(classify_level=basic, topic={simulator,ogdc,protocol})
_method **ogdc_protocol**.init(p_config)

_pragma(classify_level=basic, topic={simulator,ogdc,protocol})
_method **ogdc_protocol**.run()

_pragma(classify_level=basic, topic={simulator,ogdc,protocol})
_method **ogdc_protocol**.reset()

_pragma(classify_level=basic, topic={ simulator,ogdc,protocol})
_method **ogdc_protocol**.stop()

_pragma(classify_level=basic, topic={ simulator,ogdc,protocol})
_method **ogdc_protocol**.find_nodes(p_sensor)

_pragma(classify_level=basic, topic={ simulator,ogdc,protoco })
_method **ogdc_protocol**.find_next_nodes(p_j_sensor, p_k_sensor)

_pragma(classify_level=basic, topic={ simulator,ogdc,protocol})
_method **ogdc_protocol**.get_random_node()

_pragma(classify_level=basic, topic={simulator,bidding,protocol})
_method **ogdc_protocol**.get_node_closest_to(p_coord)

_pragma (classify_level=basic, topic={simulator,ccp,protocol})
_method **ccp_protocol**.init(p_config)

_pragma (classify_level=basic, topic={simulator,ccp,protocol})
_method **ccp_protocol**.run()

_pragma (classify_level=basic, topic={simulator,ccp,protocol})
_method **ccp_protocol**.reset()

_pragma (classify_level=basic, topic={simulator,ccp,protocol})
_method **ccp_protocol**.stop()

_pragma (classify_level=basic, topic={simulator,ccp,protocol})
_method **ccp_protocol**.calculate_eligibility(p_sensor)

_pragma (classify_level=basic, topic={simulator,ccp,protocol})
_method **ccp_protocol**.get_union(p_neighbours)

B.3 Database Exemplars

_pragma(classify_level=basic, topic={simulator,map_objects})

```
sensor.define_shared_variable( :area_rad, 1, :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
sensor.define_shared_variable( :comm_rad, 1, :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
sensor.define_shared_variable( :inaccurate_offset, 0, :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
sensor.define_shared_variable( :neighbours, {}, :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
sensor.define_shared_variable( :movement_table, hash_table.new(), :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.movement_vector

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.movement_vector << p_value

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.deploy( _optional p_area_rad, p_comm_rad )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.clean()

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.activate()

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.deActivate()

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.set_sensor_range( _optional p_new_area )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.set_communication_range( _optional p_new_comm )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.set_location( p_new_coord )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.move_to( p_new_coord, p_plot_trail? )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.reset()

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.get_distance_traveled()
```

```
_pragma(classify_level=basic, topic={simulator,geometry})
_method sensor.get_coverage_hole_for( )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.insert_trigger()

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.commit()

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.calc_location

_pragma(classify_level=basic, topic={simulator,map_objects})
_method sensor.move_from_vector( _optional p_clear_rope? )

_pragma(classify_level=basic, topic={simulator,map_objects})
export_data.define_shared_variable( :s_distance_moved, rope.new(), :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
export_data.define_shared_variable( :s_coverage_hole, rope.new(), :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
export_data.define_shared_variable( :c_coverage_hole, rope.new(), :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
export_data.define_shared_variable( :c_active, rope.new(), :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
export_data.define_shared_variable( :c_mobile, rope.new(), :public )

_pragma(classify_level=basic, topic={simulator,map_objects})
_method export_data.init()

_pragma(classify_level=basic, topic={simulator,map_objects})
_method export_data.reset_data()
```