CHAPTER III

METHODOLOGY

3.1 Introduction

This chapter discusses the novel mathematical formulation and methods that have been employed in the development of the single fitness function analysis for energy-consumption and radio bandwidth management to coverage area problems. The chapter discusses the theoretical formulation of the model, simulation approach as well as implementation procedure used. The simulator was developed utilizing C# on Microsoft Visual Studio.NET 2010 platform.

3.2 Single Fitness Function

The use genetic algorithms as the optimization tool of the developed system and an appropriate fitness function is developed to incorporate many aspects of network performance. The design characteristics optimized by the genetic algorithm system include the status of sensor nodes (whether they are active or inactive), network clustering with the choice of appropriate clusterheads and finally the choice between two signal ranges for the simple sensor nodes. The showed that optimal sensor network designs constructed by the genetic algorithm system satisfy all application-specific requirements, *fulfill* the existent connectivity constraints and incorporate energy-conservation characteristics. Energy management is optimized to guarantee maximum life span of the network without lack of the network characteristics that are required by the specific application.

Furthermore, single fitness function weighting the meaning a functions that can solve problem. In the case under investigation the fitness function is a weighting function that measures the quality and the performance of a specific sensor network design. This function is maximized by the GA system in the process of evolutionary optimization. A fitness function must include and correctly represent all or at least the most important parameters that affect the performance of the WSN design. Having described these parameters, the next issue is the decision on the importance of each parameter on the final quality and performance measure of the network design. The final form of the weighting linear fitness function f of a specific WSN design is given by to following equation.

$$f = 1/(\alpha_1.MRD + \alpha_2.SDE + \alpha_3.SCE + \alpha_4.SORE + \alpha_5.OE + \alpha_6.CE + \alpha_7.BCP) \quad (1)$$

Where;

MRD = Mean relative deviation

SDE = Spatial density error

SCE = Sensors-per-CH error

SORE = Sensors out of range

OE = Operational energy

CE = Communication energy

BCP = battery capacity penalty

The significance of each parameter is defined by setting appropriate weighting coefficients α_i = 1,2,...,7 in the fitness function that will be maximized by the GA. The values of these coefficients are determined based on experience about the importance of each parameter. First, weighting coefficients that resulted, in average the same importance of each parameter are determined first column of this table and

after some rudimental experimentation, the final values that best represented the intuition about relevant importance of each parameter are set as the second column of Table1. As can be seen in Table1, the final weights are such that network connectivity parameters weights (α_3, α_4)are treated as constraints, in the sense that all sensors should be in range with a cluster head and no cluster head should be connected to more than the predefined maximum number of sensors. There is no need for an increase of the SDE weight value.

Table1: Weighting coefficients of GA fitness function:

Weighting Coefficients	Equal Importance Values	Final Values
α_1	10^2	10^2
α_2	10^4	10^4
α_3	2	10^6
α_4	10^3	10^5
α_5	10	10
α_6	$5 * 10^{-3}$	10^{-2}

That specific constraint the desired spatial density of measurement points. Note that the coefficients are determined based on normalization with respect to the value of α_5 which is set equal to 10. It should be noted that the BCP parameter is not taken into account in the optimization of the initial design of the WSN, as it is assumed that all sensor nodes had full battery capacities at the beginning. The final value of α_7 is the result of a trade-off between energy management optimization and network characteristics optimization, particularly of the characteristics concerning the application-specific properties of the WSN, as it is further explained in Section.

There is proposing an algorithm to dynamically design WSN topologies by optimizing energy-related parameters that affect the battery consumption of the sensors and thus, the life span of the network.

At the same time, the proposed algorithm tries to meet some embedded connectivity constraints and optimize some physical parameters of the WSN implemented by the nature of the specific application. The multiple objectives of the optimization problem are blended into a single objective function, the parameters of which are combined to formulate a fitness function that gives a quality measure to each WSN topology and it is optimized by the proposed algorithm, as it is shown in following Section.

There identify three sets of parameters which dominate the design and the performance of a WSN for precision agriculture. The first set is the application specific parameters which include two parameters regarding the deployment of sensors for the specific case considered here. These are the highest possible uniformity of sensing points and some desired spatial density of measuring points. The second set is the connectivity parameters which include an upper bound on the number of sensors that each clusterhead sensor can communicate with, and the fact that all sensors must have at least one clusterhead within their signal range. Finally, the third set refers to the energy-related parameters which include the operational energy consumption depending on the types of active sensors, the communication energy consumption depending on the distances between sensors that communicate with their corresponding clusterhead, and finally the battery energy consumption.

The optimization problem is defined by the minimization of the energy-related parameters see Table 2 and the maximization of sensing points' uniformity, subject to the connectivity constraints and the spatial density requirement (see Table for the

exact correspondences). In order to combine all objectives into a single objective function (weighted sum approach), the optimization parameters are formed in such a way that all of them are minimized. Thus, objective j4 is expressed by its dual objective; say j'4, which has to be minimized. Further, the penalization of the constraints allows their transformation into objectives j5, J6, and j7, respectively, which have to be minimized. Thus, a single objective function that blends all (obviously conflicting) objectives is of the form

$$f = \min\left\{ \sum_{\substack{i=1 \\ i\neq 4}}^{7} w_i \, j_i + w_4 \, j''_4 \right\} \qquad (2)$$

This form of objective function is suitable for the formulation of a numeric evaluation function by Michalewicz. Z, D.B,. (2002) (namely a ''fitness function'' in the terminology of GAs), which gives a quality measure to each possible solution of the optimization problem. The details of that formulation are presented in Section. What follows describes the mathematical representation of the optimization parameters in their ''minimization'' form.

Table 2: Correspondences between objectives and optimization parameters

Objectives	Optimization Parameters	Parameter Symbols in GA methodology
j_1	Operational energy	OE
j_2	Communication energy	CE
j_3	Battery capacity penalty	BCP
j_4	Uniformity of measurement	
$j_4^{\bullet}$	Mean relative deviation of measurement points	MRD
j_5	Sensors-per-CH error	SCE
j_6	Sensors out of range	SORE
j_7	Spatial density error	SDE

The application-specific parameters: The main goal of a WSN used in precision agriculture is to take uniform measurements over the entire area of interest, so that an overall and uniform picture of the conditions of the area is realized. This has been achieved using the following two parameters.

A- First, the measure of uniformity of measurements. The metric of the uniformity of measurement points that was used here was the Mean Relative Deviation (*MRD*). The entire area of interest was divided into several overlapping sub-areas. Sub-areas are defined by four factors: two that define their size (length and width) and two that define their overlapping ratio (ratios in the two directions). All these factors are expressed in terms of the unit length of each direction. The larger the overlapping ratio is, the higher

precision is achieved in the evaluation of uniformity, but also, the slower the algorithm becomes.

In order to define *MRD*, the notion of spatial density (ρ) of measurements is used. More specifically, ρ_{si}, the spatial density of measurements in sub-area s_i, is defined as the number of measurements over the area of the *i*th sub-area, *i*=1,2,...,N, where N is the number of overlapping sub-areas into which the entire area, say S, is divided. In addition, ρ_s, the spatial density of the entire area of interest, is defined as the total number of measurements of the network over the total area of interest. Thus, *MRD* is defined as the relative measure of the deviation of the spatial density of measurements in each sub-area from the total spatial density of measurements in the entire area

$$MRD = \frac{\sum_{i=1}^{N} |\rho_{si} - \rho_s|}{N.\rho_s} \tag{3}$$

Where;

MRD = Mean relative deviation

N= number of overlapping

ρ_i =The spatial density of measurements in sub-area

ρ_s = The spatial density of the entire area of interest

Low values of *MRD* mean high uniformity of measurement points. Acceptable values for my application example are of *MRD* below 0.15.

B- The second application-specific parameter of the fitness function was the Spatial Density Error (*SDE*) that was used to penalize network designs that did not meet the minimum required spatial density of measurement points that

would suffice adequate monitoring of the measured variables (air or soil temperature, air or soil relative humidity, solar radiation) in the area of interest. The desired spatial density ρd is set equal to 0.2 measurement points

$$SDE = \begin{cases} \dfrac{pd - p^s}{p^d} & if p_s \succ pd \\ 0 & otherwise \end{cases} \quad (4)$$

per square length unit and the *SDE* factor is evaluated by

Where;

SDE= Spatial density error

ρd = The desired spatial density ρd is set equal to 0.2 measurement points per square length unit and the SDE factor is evaluated.

ρ_s = The spatial density of the entire area of interest

In connectivity parameters a crucial issue in WSNs is the assurance that network connectivity exists and all necessary constraints are satisfied. Here, these necessary characteristics of the sensor network are taken into account by the inclusion of the following parameters in the fitness function:

a- A Sensors-per- Clusterhead Error (SCE) parameter to ensure that each clusterhead did not have more than a maximum predefined number of sensors in regular operating modes in its cluster. This number is defined by the physical communication capabilities of the sensors as well as their data management capabilities in terms of quantity of data that can be processed by a clusterhead sensor. It was assumed to be equal to 15 for the application considered here. If *nfull* is the number of clusterheads or clusters that has more

than 15 active sensors in their clusters and n_i is the number of sensors in the ith of those clusters, then

$$SCE = \begin{cases} \dfrac{\sum_{i=1}^{\inf ull} n_i}{nfull} if & nfull \\ 0 & otherwise \end{cases} \succ 0, \quad (5)$$

Where;

SCE= Sensors-per-CH error

nfull = the number of clusterheads (or clusters) that have more than 15 active sensors in their clusters

n_i = The number of sensors in the ith of those clusters

b- A Sensors -Out- of-Range Error (SORE) parameter to ensure that each sensor can communicate with its clusterhead. This of course depends on the signal range capability of the sensor. It is assumed that HSR-sensors cover a circular area with radius equal to 10 length units, while LSR-sensors cover a circular area with radius equal to 5 length units. If *nout* is the number of active sensors that cannot communicate with their clusterhead and n is the total number of active sensors in the network, then

$$SORE = \frac{nout}{n} \quad (6)$$

Where;

SORE= Sensors-Out-of-Range Error

nout= The number of active sensors that cannot communicate with their clusterhead

n= the total number of active sensors in the network

Energy-related parameters: Energy consumption in a wireless sensor network, as explained earlier, is a crucial factor that affects the performance, reliability and life span of the network. In the optimization process during the evolutionary design of the sensor network, three different energy-related parameters are taken into account:

a- The operational Energy (OE) consumption parameter, which refers to the energy that a sensor consumes during some specific time of operation. It basically depends on the operation mode of the sensor, that is, whether it operates as a CH, a HSR or a LSR sensor, or whether it is inactive. The corresponding relevance factors for the energy consumption of the three active operating modes of the sensors are taken proportional to 20:2:1, respectively and zero for inactive. The meaning is that the energy consumption of a sensor operating in CH mode is 10 times more than that of a sensor operating in HSR mode and 20 times more than that of a sensor operating in LSR mode. These relevant factors are used to simplify the analysis and did not necessarily represent accurately the real energy relations between the available operation modes of the sensors. Their exact values depend on electromechanical characteristics of the sensors and are not further considered in the analysis presented here. The *OE* consumption parameter was then given by

$$OE = 20.\frac{noh}{n} + 2.\frac{nhs}{n} + \frac{nls}{n} \tag{7}$$

where;

OE= Operational energy

where, *nch, nhs* and *nls* are the number of CH, HSR and LSR sensors in the network, respectively.

b- Communication Energy (CE), which refers to the energy consumption due to communication between sensors in regular operating modes and clusterheads. It mainly depends on the distances between these sensors and their corresponding clusterhead, as defined by in. Ghiasi S, A. Srivastava, X. Yang, M. Sarrafzadeh (2002). It is depicted by

$$CE=\sum_{i=1}^{c}\sum_{j=1}^{n_i}\mu.d_{ji}^{k} \qquad (8)$$

Where;

CE=Communication energy

where c is the number of clusters in the network, n_i is the number of sensors in the *i*th cluster, d_{ji} is the Euclidean distance from sensor *j* to its clusterhead (of cluster *i*) and μ and *k* are constants, characteristic of the topology and application site of the WSN. For the specific precision agriculture application for open field monitoring, the values $\mu = 1$ of and $k= 3$ are chosen.

c- The battery life, an important issue in WSNs is self-preservation of the network itself, that is, the maximization of the life span of the sensors. Each sensor consumes energy from some battery source in order to perform its vital operations, like sensing, communication, data aggregation if the sensor is a clusterhead. Battery capacity of each sensor of the network was taken into account in the design optimization process by the introduction of a Battery Capacity Penalty (*BCP*) parameter. Since the operation mode of each sensor is

known, its Battery Capacity (*BC*) can be evaluated at each time. Thus, when the design optimization algorithm is applied at a specific time *t* (measuring cycle), the *BCP* parameter is given by

$$BCP^{[t]} = \sum_{i=1}^{ngrid} PF_i^{[t]} \left(\frac{1}{BC_i^{[t]}} - 1 \right), t = 1,2, \quad (9)$$

Where;

BCP= Battery capacity penalty

BC= Battery Penalty

PF= Penalty Factor

Note that BC_i is updated according to the operation mode (CH, HSR or LSR) of each sensor *i*, during the previous measuring cycle *t*-1 of the network

$$BC_i^{[t]} = BC_i^{[t-1]} - BRR_i^{[t-1]}. \quad (10)$$

where;

BC= Battery Capacities

BRR= Battery Reduction Rate

In the above:

1. $BCP^{[t]}$ is the Battery Capacity Penalty of the WSN at measuring cycle *t*. It is used to penalize the use of sensors with low battery capacities, giving at the same time larger penalty values to operating modes that consume more energy (especially CH mode).

2. Ngrid is the total number of available sensor nodes.

3. $PF_i^{[t]}$ I is the Penalty Factor assigned to sensor *i*. The values it takes are given according to the operation mode of sensor *i*. The values used here are proportional to

the relevant battery consumptions of the sensor modes, namely, 20:2:1 for active sensor modes (CH, HSR and LSR, respectively) and 0 for inactive. They provide different penalties according to the specific modes of the sensors in the WSN of the following measuring cycle. However, as it is explained in the next section, further exploration of the optimal relevance values needs to be performed.

4. $BC_i^{[t-1]}$ and $BC_i^{[t-1]}$ are the Battery Capacities of sensor i at measuring cycles t and t-1, respectively, taking values between 0 and1, with 1 corresponding to full battery capacity and 0 to no capacity at all.

5. $BRR_i^{[t-1]}$ is the Battery Reduction Rate that depends on the operation mode of sensor i during the measuring cycle t-1 and reduces its current battery capacity accordingly, using the percentage of the relevance factors for the energy consumption of the operating modes of the sensor as follows: 0.2 for CH, 0.02 for HSR 0.01 for LSR operation modes and 0 for inactive sensors.

Having completed the development of a representation scheme and forming the single fitness function, the dynamic genetic algorithm for optimal adaptive design of the WSN could be developed by controlling algorithm for energy-consumption and user bandwidth in WSN using single fitness function.

Conclusions, the algorithm for the optimal design and dynamic adaptation of application-specific WSNs, based on the evolutionary optimization properties of genetic algorithms as presented. A fixed wireless network of sensors of different operating modes is considered on a grid deployment and the GA system decided which sensors should be active, which ones should operate as clusterheads and whether each of the remaining active normal nodes should have high or low-signal

range of saving. During optimization, parameters of network connectivity, energy conservation as well as application requirements are taken into account so that an integrated optimal WSN was designed.

From the evolution of network characteristics during the optimization process, which can conclude that it is preferable to operate a relatively high number of sensors and achieve lower energy consumption for communication purposes than having less active sensors with consequently larger energy consumption for communication purposes. In addition, GA-generated designs compared favourably to random designs of sensors. Uniformity of sensing points of optimal designs is satisfactory, while connectivity constraints are met and operational and communication energy consumption is minimized. That also showed that the dynamic application of the algorithm in adaptive WSN design can lead to the extension of the network's life span, while keeping the application-specific properties of the network close to optimal values. The algorithm showed sophisticated characteristics in the decision of sensors' activity/ inactivity schedule as well as the rotation of operating modes (clusterhead or ''regular sensor'' with either high or low-signal range), which led to considerable energy conservation on available battery resources.

For future work, will deal with the development of heuristic methodologies for optimal routing of dynamically selected clusterhead sensors, through some multi-hop communication protocol will be death with

3.3 Simulation of the Wireless Sensor Network

In order to achieve the aim of this research, a simulation application of the wireless sensor network was developed. A number of parameters determine the outcome of the

simulation depending on the deployed strategy adopted in the simulation. These parameters include relative network size or number of nodes, distance between communication nodes, energy requirement to transmit or receive packets. Depending on the parameters configuration, the network is then utilised to model performance of the setup to determine the vectors travelling across the sensor network field. The approach here is focused on the coverage area & energy optimisation. The intention is to examine what happens when the vector trips the sensor of a network node. Usually, the node generates a data packet and then forwards it to a downstream network node. The packets are then routed randomly based on the configuration of the sensor network until they reach a sensor within the "uplink zone". Each node has an energy store that is spent as packets are received or forwarded as well as in detecting vectors. Of course, since the energy at the nodes are finite, it is a matter of time for them to run down and become dead in the sensors network, eventually leading to network failure.

This simulation setup is able to model and run consecutive tests on a network. From these runs, the value of mean network lifetime across 1,000 trials can be acquired. Changing the network routing parameters to setup different network configurations allows variations in the setup and proper benchmark to determine optimal values.

3.4 Simulation Setup

The simulation reported in this work can be divided into two stages: deploying the network and running simulations.

The first step in deploying the network is to set the parameters of the network utilising the configuration sliders. These parameters must be set on the onset when the

network is created. New modification to the network configuration and routing parameters cannot take place unless a new network is deployed. The parameters of a network configuration can be classified into two categories:

3.4.1 Network Configuration

This group of parameters concerns determine the physical configuration of the network, namely, hardware properties of the network. These include the following variables:

- *Network Size*: The total figure: 10,15,20,25,30,35,40,45,50,55. number of nodes to be deployed in the network. Setting the network size to a high value implies having several hundred nodes in the network. With such a value, there will be a huge density of network connections and network becomes heavy. This eventually will have heavy impact on the simulation performance. However, sometimes if there is the need for particularly large networks, then the best approach is to reduce the Transmission Radius.

- *Sensor Radius*: The sensor radius is the proximity distance between the Sensors in the network. The value of the ten (10) sensor radius starts from 24 to 119. The values are 24, 45, 61, 73, 75, 76, 76, 81, 105, and 119.

- *Sensor Cost*: This is a measure of the energy required cost in detecting a vector and generating a packet. The value of then ten (10) of sensor cost begin From 20 to 72. The values are 20, 43, 45, 47, 62, 63, 64, 65, 69, and 72.

- *Transmission Radius*: The transmission radius refers to the maximum distance possible for two network nodes to see each other for communication.

Setting this very high means that the nodes very far apart may be able to communicate, even up to those on opposite sides of the map. Likewise, setting it to a low value implies that those nodes closer together are the ones that would be able to communicate. The value of then ten (10) transmission radius 130.

- *Transmitter Period*: This is the time required to send a packet. A high value indicates that packet transmission will take several seconds. This thereby makes the data delivery to take longer time before delivery at the radar due to the elapsed time since the triggering event. However, setting it to a high period gives the user sufficient room to monitor the packet-exchange process on the network map. . The values of then ten (10) transmitter period start from 10 to 67. The values are 10, 42, 43, 45, 52, 52, 53, 54, 64, and 67.

- *Transmit Cost*: This is the energy cost expended in sending a packet. A very high value leads to a rapid depletion of a node's energy during transmission, which invariably leads to wearing out after sending only a few packets. Setting this value very low on the other hand implies that the nodes may be able to send several hundred of packets. However, it should be noted that transmit cost is scaled based on the distance between the nodes. Therefore, energy is often rapidly depleted since more distant nodes can only be reached by a more powerful broadcast. The value of then ten (10) transmits cost start from 2 to 67.75. The value are 2 , 2 , 35.49 , 39.79 ,47.32 , 52,7 , 59.15 , 62.38 , and 67.75.

- *Receive Cost*: This is the energy cost in receiving a packet. Once, it takes the value of the transmit cost once the transmit cost is set. There is no need to

particularly scale it. The values of then ten (10) receive cost start from 15 to 81. The values are 15, 46, 55, 60, 63, 64, 70, 70, 73 and 81.

Table 3 Experiment 1.

Number	Network size	Transmitter delay
1	10	10
2	15	45
3	20	43
4	25	54
5	30	52
6	35	52
7	40	53
8	45	64
9	50	42
10	55	67

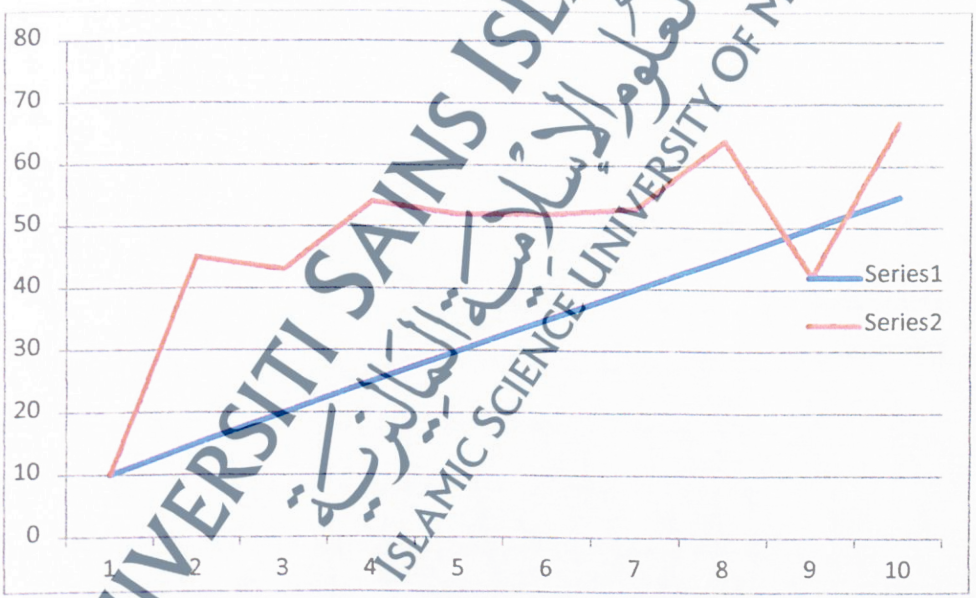

Figure12: Experiment 1

Conclusion, if the number of network size start from 15 to 45, then it will almost be a linear increment in the transmitter delay.

Table 4 Experiment 2.

number	Network size	Transmission cost
1	10	2
2	15	59.15
3	20	2
4	25	47.32
5	30	48.4
6	35	35.49
7	40	39.79
8	45	52.7
9	50	62.38
10	55	67.75

Figure13: Experiment 2

Conclusion, if the number of network size start from 30 to 45, then it will almost be a linear increment in the sensor cost.

Table 5 Experiment3.

Number	Network size	Sensor cost
1	10	20
2	15	64
3	20	69
4	25	45
5	30	43
6	35	62
7	40	65
8	45	63
9	50	47
10	55	72

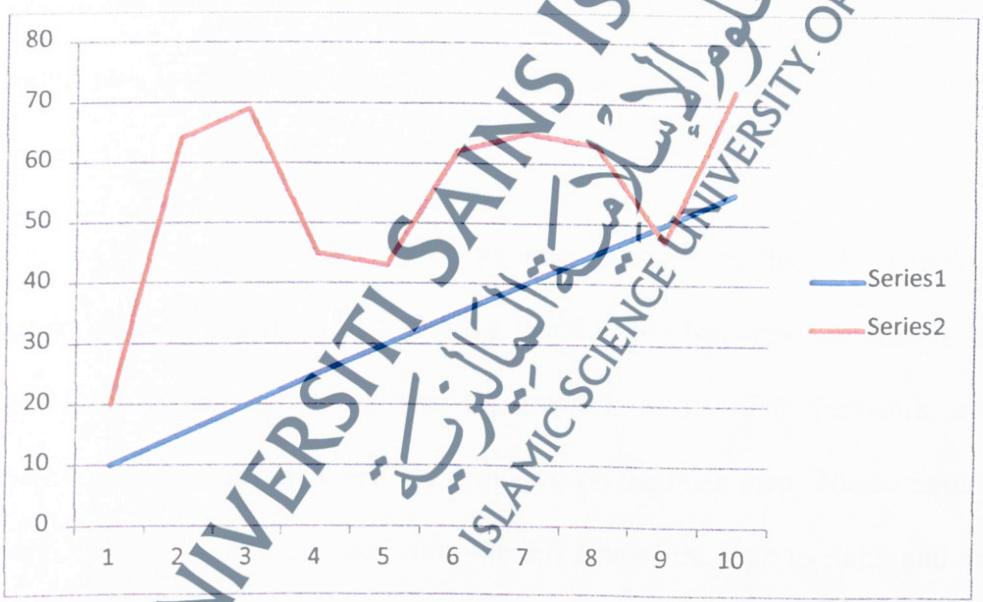

Figure14: Experiment 3

Conclusion, if the number of network size start from 35 to 55, then it will almost be a linear increment in the transmission cost.

There are seven (7) parameters in the experiment. The network size parameters will be the main comparison standard in the conclusion. The comparison will be done using network size as the main parameter and it will be compared with the other six (6) parameters. For each comparison the best trend of comparison will be selected.

3.4.2 Routing Parameters

These include some other configuration setting required. The factors included in this category indicate software properties of the network needed for the packet-routing method. It is often either set to "Random" or "Directed". Setting routing to "Random," means that each node selects a downstream connection randomly for each packet while a "Directed," routing utilises a specific algorithm for the packets' routing specifically based on the algorithm (Chang and Taissulas, 1999). The directed routing parameters include Exchange Cost, Residual Energy, Initial Energy, and Routing Period.

Once the outlined network parameters are specified, the simulation of the network can be deployed by activating the "Deploy Network" command. On clicking the "Deploy Network" to activate the network, nodes of the network are randomly distributed and connected, and are displayed on the main map. The communications of the network are often initiated from the left side of the map to right, and nodes in the "uplink zone" or rather the striped zone on the right side of the map are presumed to be in direct contact with the data collector. In a situation where an alternative random distribution of nodes is desired, the "Deploy Network" can be activated again by clicking its command button.

Subsequently, after the network has been deployed, then clicking the "Start Simulation" button activates the simulation process. The distribution of the vectors moving through the field and triggering sensors is displayed on the map. In the scenario created, the sensors energy is expended and over time, some of the sensors in the network may run out of power, thereby dropping out of the network. Eventually, the energy at each node may run down and whole network is powered down.

Using the "Simulation Status" window, the progress of the network can be observed and monitored and a new simulation configuration may be initiated by ending a previous setup and restarting the new simulation setting. Alternatively, the previous simulation may be reviewed by clicking the "Replay Simulation" button.

3.4.3 Display

The network configuration can be visualised on the main map, often as a series of red circles delimited by gray circles. The red circle depicts the sensor or node, while the gray circles signify the area surrounding the node forming the sensor detection range. The vector is depicted with the moving green rectangles. If any vector enters this area the sensor is triggered. As a result, the gray area turns bluish which gradually fades until it becomes gray. The speed with which this fading occurs is dependent on the sensor delay period.

Nodes are connected to other nodes using black lines, which signify the communications links. In directed routing configuration, present connections are indicated with colour blue. Any active connection that is exchanging a packet appears red. Often this is not really visible since the more realistic low transmission period will not guarantee prolong visibility. This is because packets are transmitted so rapidly

making the line appears red only briefly. However, if the transmitter period is set to a high value, it may be more visible due to being a little prolong.

The battery colour depicted at the centre of the red circle represents node's battery. The status of the node which is white coloured when in full power mode is gradually turned to complete black at the exhaustion of power. Eventually, at the complete depletion of the node's power, the red circle around the node, disappears and becomes completely black, the gray sensor area vanishes and likewise all of the communications links vanish.

The status radar at the bottom of the display screen presents the results of the data transfer. In the radar, the nodes are seen as green circles while the vectors on the other hand are depicted as some small rectangles. A packet that successfully reaches a node in the uplink zone of the network is transferred to the radar and displayed as a hit changing the colour of the circle to bright green. This allows the monitoring the speed and accuracy of the network regarding how the vectors pass through the field.

3.4.4 Programming

The application development is in two modules. The first module is the development of the wireless sensor network while the other module was the creation of the simulator that hosts the wireless sensor network objects. The descriptions of the classes that comprise the wireless sensor network are described below:

* WirelessSensor: This class is used to identify a single sensor. Some basic parameters are contained in this class, namely, iSensorRadius (the radius, in pixels, of sensitivity of the sensor), iSensorDelay (set to zero if the sensor is ready for detection, or positive if the sensor has recently detected a vector and is momentarily

desensitized.), and x and y (integers representing the position of the node on the map.)

Noteworthy member variables include:

* aPackets: an ArrayList of Packet objects, each representing a datagram to be forwarded to one of the downstream connections.

* aConnections: an ArrayList of WirelessSensorConnections representing connections to downstream nodes.

* connectionCurrent: if directed routing is in use, the currently selected downstream node.

* iResidualEnergy: the amount of battery power remaining in the sensor and node; the sensor is disabled when this falls to or below 0.

* WirelessSensorConnection: This class represents a connection between two network nodes, internally designated as sSender and sReceiver. The connection contains a placeholder for the packet being exchanged, as well as a timer to simulate a non-instantaneous exchange period. These objects fit into the simulation as members of the aConnections list (always hosted by the upstream node.)

* Packet: This class represents the datagram exchanged between nodes. It simply contains variables x, y, lifetime (the maximum amount of time that the sensor will appear on the radar), and timestamp (the amount of time left until the packet expires on the radar.)

* WirelessSensorNetwork: This class represents the entire network. In addition to many parameters that correspond to the sliders in the simulator (Receiver Cost, Sensor Cost, Sensor Delay, etc.) and some data synchronization variables, this class contains

two important ArrayLists: aSensors, which holds all of the WirelessSensor objects, and aRadar, which holds all of the packets that have been transmitted out of the network and are appearing on the radar.

* VectorList and Vector: The VectorList class contains a list of Vector objects, each of which represents an object moving across the map and being detected by the network sensors.

The interface class is a typical Windows .NET event-driven interface. Only two aspects of this system are noteworthy:

* The actual network simulation runs on a separate thread from the thread controlling the user interface. This simulator supports completely gigantic networks – 400 nodes x 400 nodes, which, fully interconnected (maximum network transmission radius), include 160,000 network connections. Simulating transmissions across all of these connections, and then drawing the whole network several times per second, requires a nontrivial deal of processing. The CPU of a typical 3.2GHz machine simply can't juggle this task with the message pump that handles window events; hence, cramming both functions into one thread results in a completely unresponsive simulator window. Instead, the simulation thread is started in a separate thread, which runs continuously until the user stops it (by clicking Stop) or closes the window; as a result, the window remains responsive during the simulation.

* Isolating the drawing and simulation functions to separate threads creates a synchronization problem: If the main thread is tracing the Array List of vectors at the same time that the processing thread updates the list (by inserting or, worse, removing

an item), there exists the likelihood of an exception. To preclude this occurrence, the threads synchronize access to the vector list by using a mutex.

3.4.5 Explaining of source codes in simulator.

This class represents the entire wireless sensor network. The network data structures. The number of nodes of the network total figure: 10,15,20,25,30,35,40,45,50,55, nodes. The initial energy of every node 1000 Joules. And also the max X dimension of the map for randomly placing nodes and max Y dimension of the map for randomly placing nodes. The location of the "end zone" (data collection uplink). The pixel radius of a sensor for detecting a vector. The duration of the delay between packets sent by a tripped sensor. The duration of the packet transmission. The maximum communications range of any network node. The percent energy cost of sending a packet across (x) pixels - scales with distance, to be realistic. The energy cost of receiving a packet of information.

When the energy cost of a tripped sensor. That can showed specifies whether routing is directed or random. The lifetime of directed routing information that will focus in further work. Depended in how long the network waits before re-routing. The directed-routing network parameters. Network simulation variables, specifies whether or not the visual simulation is running. And also specifies whether or not the map should be painted, and whether or not it is currently being painted. Sepcifies whether or not the simulation thread should be aborted. Specifies whether or not endurance testing is occurring. In addition, the amount of time the network has been running (solely useful for visual simulation). A timer for updating routing information. The number of cycles executed by the process (solely useful for endurance testing). The number of packets delivered by the network. The simulation thread (set to NULL

when no simulation is running). A random number producer. total figure: 10,15,20,25,30,35,40,45,50,55. The last seed used (retained for the purpose of replaying a simulation, by re-seeding with this value). The results of each test (100-integer array). This is, admittedly, a crude way to write a constructor - should be broken out into a network descriptor class - but hey, it works.

Build the network simulation function, first, add random sensors to the network (keep sorted by x, and make sure no other nodes with this x/y coordinate are in the list). Add to list - note that it's sorted according to X coordinate; this provides an unambiguous metric for establishing routing connections (from nodes earlier in the list to nodes further down the list). Not added to list - add at the end. Establish connections (wipe out all sensor connection grids). Finally, weed out all sensors with no downstream connections that aren't in the destination zone. Furthermore, the trick here is to ensure that there aren't any dead spots in the network, i.e., clusters of nodes that are connected to each other but not to a node in the Data collector/ uplink zone - that do this by removing all nodes that have no downstream connections and aren't in the uplink zone.

Wireless Sensor sensor = (WirelessSensor)aSensors[i];if ((sensor.x < iDestinationX) && (sensor.aConnections.Count = = 0))

Dead end - eliminate. The scan all upstream nodes, to see if they were connected to this removed node, and delete the connections.

WirelessSensor sensor2 = (WirelessSensor)aSensors[j];

```
                ArrayList aRemoveConnections = new ArrayList();
                foreach (WirelessSensorConnection connection in
sensor2.aConnections)
                {
                  if (connection.sReceiver == sensor)
                    aRemoveConnections.Add(connection);
                }
                foreach (WirelessSensorConnection connection in
aRemoveConnections)
                  sensor2.aConnections.Remove(connection);
              }
            }
          }
          foreach (WirelessSensor sensor in aRemoveSensors)
            aSensors.Remove(sensor);
        }
      }

      public void Reset(bool bNewSeed)
      {
```

This function resets the network so that a new simulation can be run - can either be reset with a new seed, or with the previous seed (for replay.).

```
        this.iProcessTime = 0;
        this.iPacketsDelivered = 0;
        foreach (WirelessSensor sensor in aSensors)
        {
          sensor.iResidualEnergy = sensor.iInitialEnergy;
          sensor.aPackets = new ArrayList();
          sensor.iSensorRadius = iSensorRadius;
          sensor.iSensorDelay = 0;
```

```
        foreach (WirelessSensorConnection connection in sensor.aConnections)
        {
            connection.iTransmitting = 0;
            connection.packet = null;
        }
    }
    aRadar = new ArrayList();
    if (bDirectedRouting == true)
        SetRoutingInformation();
    iLastUpdated = iUpdateDelay;
    if (bNewSeed == true)
        this.iSeed = (int)System.DateTime.Now.Ticks;
    r = new Random(iSeed);
}

public void RunSimulation()
{
```

This function is the heart of the visual simulation thread. It runs until b Abort is set to true - this is a faster and cleaner stop mechanism than using thread.Abort().

```
bAbort = false;
    bRunningSimulation = true;
    timeStart = System.DateTime.Now;
    this.vectors = new VectorList(iMaxX, iMaxY, r);
    while (bAbort == false)
    {
        Process(true);
        System.Threading.Thread.Sleep(20);
```

```
    }
    bRunningSimulation = false;
}

public void RunTest()
{
```

This function showed the runs the endurance test.

```
    bTesting = true;
    iTestResults = new int[100];
    bAbort = false;
    bRunningSimulation = true;
    timeStart = System.DateTime.Now;
    this.vectors = new VectorList(iMaxX, iMaxY, r);
    for (int i = 0; i < 100 && bAbort == false; i++)
    {
        Reset(true);
        while (Process(false) == true)
            iTestResults[i]++;
    }
    bRunningSimulation = false;
    bTesting = false;
}

public bool Process(bool bSleep)
{
```

This is the core of a network simulation - all of the network processes for one step of the simulation occur here. Returns a bool to indicate that the. Network is alive (all nodes active) or dead (partitioned by one or more dead nodes.).

```
bool bNetworkLives = true;
        iProcessTime++;
        // update vectors (note use of mutex)
        vectors.mutexVector.WaitOne();
        vectors.Update();
        while (vectors.aVectors.Count < 3)
          vectors.AddVector();
        vectors.mutexVector.ReleaseMutex();
```

The update of sensors networks (creates a new packet if sensor is active and triggered by a vector).

```
foreach (WirelessSensor sensor in aSensors)
        {
          if (sensor.iResidualEnergy > 0)
          {
            if (sensor.iSensorDelay > 0)
              sensor.iSensorDelay--;
            else
            {
              foreach (Vector vector in vectors.aVectors)
              {
                if (Math.Sqrt(Math.Pow(vector.x - sensor.x, 2) + Math.Pow(vector.y
- sensor.y, 2)) < sensor.iSensorRadius)
                {
                  if (sensor.iResidualEnergy <= iSensorCost)  // sensing failed due
to sensor cost - sensor is now dead
```

```
                    sensor.iResidualEnergy = 0;
                else
                { // create packet
                    sensor.iSensorDelay = iSensorDelay;
                    sensor.iResidualEnergy -= iSensorCost;
                    sensor.aPackets.Add(new Packet(sensor.x, sensor.y,
sensor.iSensorDelay));
                }
            }
        }
    }
    else
        bNetworkLives = false;
}
Start new transmisions for any pending packets.
foreach (WirelessSensor sensor in aSensors)
{
    if (sensor.aPackets.Count > 0)
    {
        if (sensor.aConnections.Count > 0)
        {
            if ((bDirectedRouting == true) && (sensor.connectionCurrent != null))
                sensor.connectionCurrent.BeginTransmission();
            else  // random routing

((WirelessSensorConnection)sensor.aConnections[r.Next(sensor.aConnections.Count)
]).BeginTransmission();
        }
    }
```

```
}
// carry out transmissions across each connection
foreach (WirelessSensor sensor in aSensors)
{
   foreach (WirelessSensorConnection connection in sensor.aConnections)
   {
      Packet packet = connection.Transmit();
      if (packet != null)
      {
         aRadar.Add(packet);
         iPacketsDelivered++;
      }
   }
}
// if using directed routing, update connection information
if (bDirectedRouting == true)
{
   if ((--iLastUpdated) <= 0)
   {
      iLastUpdated = iUpdateDelay;
      SetRoutingInformation();
   }
}
```

Note the end of the process - tell the map to paint itself, and sleep for 10ms if this is a timed simulation.

```
if (bPainting == false)
   bPaint = true;
if (bSleep == true)
```

```
            System.Threading.Thread.Sleep(10);
        return bNetworkLives;
    }
public void SetRoutingInformation()
    {
 // this function updates directed-routing selections
        foreach (WirelessSensor sensor in aSensors)
        {
     if (sensor.iResidualEnergy > 0)
            {
       Choose best node given current conditions
            sensor.connectionCurrent = null;
            double dBestCost = 0;
            foreach (WirelessSensorConnection connection in sensor.aConnections)
            {
              if (connection.sReceiver == null)
              {
If this is an uplink connection, always select it
                sensor.connectionCurrent = connection;
                break;
              }
              else if (connection.sReceiver.iResidualEnergy > 0)
              {
                Double dCost = Math.Pow(connection.iTransmitCost, x1) *
Math.Pow(connection.sSender.iResidualEnergy, -x2) *
Math.Pow(connection.sSender.iInitialEnergy, x3) +
Math.Pow(connection.iReceiveCost, x1) *
Math.Pow(connection.sReceiver.iResidualEnergy, -x2) *
Math.Pow(connection.sReceiver.iInitialEnergy, x3);
                if ((sensor.connectionCurrent == null) || (dCost < dBestCost))
                {
```

```
                    dBestCost = dCost;
                    sensor.connectionCurrent = connection;
                }
            }
        }
    }
}
}
#endregion
}
public class WirelessSensor
{
```

This class represents a single node in a wireless sensor network

```
    #region Variables and initialization code
public ArrayList aPackets = null;
                    // an array of packets held by this node
    public ArrayList aConnections;
```

An array of connections to downstream nodes (*always* downstream, not upstream) - nodes in the data collector/uplink zone have a connection with a "null" receiver node public Wireless Sensor Connection connectionCurrent. IF using directed routing, the currently preferred network connection for transmitting.

```
public int x, y;
            // the coordinates of the node
    public int iInitialEnergy;
                        // the initial power of the node
    public int iResidualEnergy;
                        // the current power of the node
```

```
    public int iSensorDelay = 0;
                    // the timer until the sensor is ready to be tripped again
    public int iSensorRadius;
                            // the radius of this sensor
public WirelessSensor(int x, int y, int iSensorRadius)
    {
      this.x = x;
      this.y = y;
      this.iSensorRadius = iSensorRadius;
      aConnections = new ArrayList();
      connectionCurrent = null;
      iInitialEnergy = iResidualEnergy = WirelessSensorNetwork.iMaxEnergy;
    }
 #endregion
}
```

This class represents a communications link between two wireless sensors.

```
    #region Variables and initialization code
 public WirelessSensor sSender;
```

The upstream sensor

```
    public WirelessSensor sReceiver;
```

The downstream sensor - every node in the data collector/uplink zone will have a connection with a NULL sReceiver.

The packet currently being transmitted on this connection (only one at a time, Of course), Public int iTransmitCost, iReceiveCost; the energy costs of transmitting and receiving the packet public int iTransmitterDelay;

The total time this node would normally wait to complete delivery of a packet public int iTransmitting;

The timer for completing delivery of a packe public Wireless Sensor Connection (Wireless Sensor sSender, Wireless Sensor sReceiver, int iTransmi tCost, int iReceive Cost, int *i* Transmitter Delay)

```
    {
        this.sSender = sSender;
        this.sReceiver = sReceiver;
        this.iTransmitCost = iTransmitCost;
        this.iReceiveCost = iReceiveCost;
        this.iTransmitting = 0;
        this.iTransmitterDelay = iTransmitterDelay;
    }
  #endregion
#region Connection simulation functions
public void BeginTransmission()
    {
This function begins transmission of a data packet between these nodes.
        if ((sSender.aPackets.Count > 0) && (iTransmitting == 0))

        {
if (sSender.iResidualEnergy <= iTransmitCost)
transmission failed - sender has run out of energy
            sSender.iResidualEnergy = 0;
        else if ((sReceiver != null) && (sReceiver.iResidualEnergy <=
iReceiveCost))
transmission failed - receiver has run out of energy
            sReceiver.iResidualEnergy = 0;
        else
        {
```

```
success - accept the packet and start transmitting it.
            iTransmitting = iTransmitterDelay;
            sSender.iResidualEnergy -= iTransmitCost;
            packet = (Packet)sSender.aPackets[0];
            sSender.aPackets.RemoveAt(0);
            if (sReceiver != null)
               sReceiver.iResidualEnergy -= iReceiveCost;
         }
      }
   }
   public Packet Transmit()
   {
```

The total time this node would normally wait to complete delivery of a packet. The timer for completing delivery of a packet. In order, of this function begins transmission of a data packet between these nodes. The transmission failed - sender has run out of energy. The transmission failed - receiver has run out of energy. The success - accepts the packet and start transmitting it.

This function continues transmission of a previously accepted packet, and completes transmission if appropriate. Failed due to depleted energy.

```
if ((sSender.iResidualEnergy <= 0) || ((sReceiver != null) &&
(sReceiver.iResidualEnergy <= 0))) // failed due to depleted energy
 iTransmitting = 0;
      else if (iTransmitting > 0)
      {
transmission in progress
         iTransmitting--;
         if (iTransmitting == 0)
```

```
            {
    if (sReceiver != null)
                sReceiver.aPackets.Add(packet);
              else
              {
                Packet returnPacket = packet;
                packet = null;
                return returnPacket;
              }
            }
          }
          return null;
        }
  #endregion
  }
  public class VectorList
    {
```

This class represents (and manages) a list of vectors. The maximum X and Y coordinates of a vector. A random number generator (the same one used by the Wireless Sensor Network object that houses this object).

```
  #region Variables and initialization code
    public ArrayList aVectors;
  the array of vectors
      public Mutex mutexVector;
a mutex to control access to the vector list
      private int iMaxX, iMaxY;
  The maximum X and Y coordinates of a vector  private Random r;
```

a random number generator (the same one used by the WirelessSensorNetwork object that houses this object)

```
public VectorList(int iMaxX, int iMaxY, Random r)
    {
        this.iMaxX = iMaxX;
        this.iMaxY = iMaxY;
        aVectors = new ArrayList();
        mutexVector = new Mutex();
        this.r = r;
    }
#endregion
#region Vector list management functions
public void AddVector()
    {
This function creates a new vector and adds it to the vector list.

int iBorder = r.Next(4);
        if (iBorder == 0)
Add to left border, traveling right and up or down
            aVectors.Add(new Vector(-5, r.Next(iMaxY), r.Next(3) + 1, r.Next(7) - 3));
        else if (iBorder == 1)
Add to top border, traveling down and left or right
            aVectors.Add(new Vector(r.Next(iMaxX), -5, r.Next(7) - 3, r.Next(3) + 1));
        else if (iBorder == 2)
Add to right border, traveling left and up or down
            aVectors.Add(new Vector(iMaxX + 5, r.Next(iMaxY), r.Next(3) - 3,
r.Next(7) - 3));
        else if (iBorder == 3)
Add to bottom border, traveling up and left or right
```

```
            aVectors.Add(new Vector(r.Next(iMaxX), iMaxY + 5, r.Next(7) - 3,
r.Next(3) - 3));
        }
public void Update()
        {
```

Add to left border, traveling right and up or down. Add to top border, traveling down and left or right. Add to right border, traveling left and up or down. Add to bottom border, traveling up and left or right.

This function moves the vectors around, and removes those that have traveled out of bounds.

```
ArrayList aRemoveVectors = new ArrayList();
            foreach (Vector vector in aVectors)
            {
               vector.x += vector.dx;
               vector.y += vector.dy;
               if ((vector.x < -5) || (vector.y < -5) || (vector.x > iMaxX + 5) || (vector.y >
iMaxY + 5))
                  aRemoveVectors.Add(vector);
            }
            foreach (Vector vector in aRemoveVectors)
               aVectors.Remove(vector);
        }
    #endregion
    }
     public class Vector
     {
```

A lightweight class - really just a structure - for a vector.

```
#region Variables and initialization code
```

```
public int x, y, dx, dy;
    public Vector(int x, int y, int dx, int dy)
    {
      this.x = x;
      this.y = y;
      this.dx = dx;
      this.dy = dy;
    }
#endregion
}
public class Packet
  {
A lightweight class - really just a data structure - for a packet.

    #region Variables and initialization code
public int x, y;
    public int timestamp, lifespan; the amount of time left to display the packet on the radar (once it gets there), and the total time of displaying it
    public Packet(int x, int y, int timestamp)
{
    this.x = x;
      this.y = y;
      this.timestamp = 0;
      this.lifespan = timestamp;
    }
#endregion
}
}
```

The amount of time left to display the packet on the radar (once it gets there), and the total time of displaying it.

The initial results of the simulator provide promising results. There are seven (7) parameters in the experiment. The network size parameters will be the main comparison standard in the conclusion. The comparison will be done using network size as the main parameter and it will be compared with the other six (6) parameters. For each comparison the best trend of comparison will be selected .The GUI is able to provide the user with the visualization of the WSN and immediate changes in the different constraints are displayed. The main constraints are how the nodes are deployed and the appropriate size of the network. The neighbor discovery process and the impact of the number of neighbors have on the network. Furthermore, the protocol used and the effects that it has on the node's power supply are critical to the overall life of the network.

Conclusion, the algorithm is explained thoroughly and verified using simulation tests. If the number of network size start from 15 to 45, then it will almost be a linear increment in the transmitter delay. If the number of network size start from 30 to 45, then it will almost be a linear increment in the sensor cost. If the number of network size start from 35 to 55, then it will almost be a linear increment in the transmission cost. A brief history on the attempts of developing solutions for WSN coverage, energy consumption and other related issues is elaborated. A new developed method for controlling WSN main parameters (such as energy consumption, BW, signal strength and coverage) using single fitness function is proposed and tested. The algorithm is explained thoroughly and verified using simulation tests. The algorithm has been tested on experimental data and yet to be tested in reality using real WSN.

Thus, the next step is to evaluate the performance of the developed algorithm using the simulation toolkit as well as running the test in real WSN.

The algorithm showed sophisticated characteristics in the decision of sensors' activity/inactivity schedule as well as the rotation of operating modes (clusterhead or ''regular sensor'' with either high or low-signal range), which led to considerable energy conservation on available battery resources.

For future work will deal with the development of heuristic methodologies for optimal routing of dynamically selected clusterhead sensors, through some multi-hop communication protocol.

3.5 Wireless Sensor Networks Simulation Formulas:

That developed a simulator to address the problem. It includes the initial node deployment, node discovery and the routing protocols as well as power management.

3.5.1 Node Deployment: It is done both statically and randomly. To cover a large region, rather than deploying a single node at a time, a random deployment method is as below:

```
Random_Deployment( )
    for ii ← 0 to Number of Nodes
        do
            Node X Position ← Random (0, Width)
            Node Y Position ← Random(0, Height)
        while Grid[Node X Position, Node Y Position] != -1

        Grid[Node X Position, Node Y Position] ← ii
        Node ID ← ii
```

3.5.2. Node Discovery: This protocol broadcast a "Hello" message, if a node is within the broadcasting range, it would acknowledge the message and send back a response. The node that initiated the message will then add the acknowledging node to a list of neighbors. When a node does an initial broadcast and receives no acknowledgement, the node will increase the broadcasting range till the minimal amounts of neighbors are discovered. The following algorithm is being used:

```
Node_Discovery ( )
        foreach Node
                x ←Node X Position
                y ←Node Y Position
                for a ←  -range do until a > range
                        for b ← -range do until b > range
                                if INBOUND
                                        AND Grid[x-a,y-b] != -1
                                        AND distance (x, y, x-a, y-b, range)
                                        then Node.Neighbor ← Grid[x-a, y-b]
```

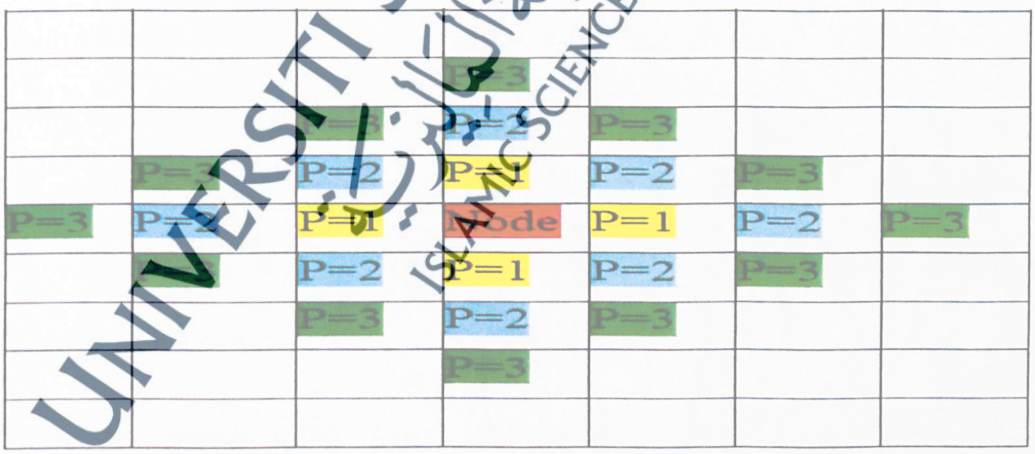

Figure 15.Von Neumann Neighborhood

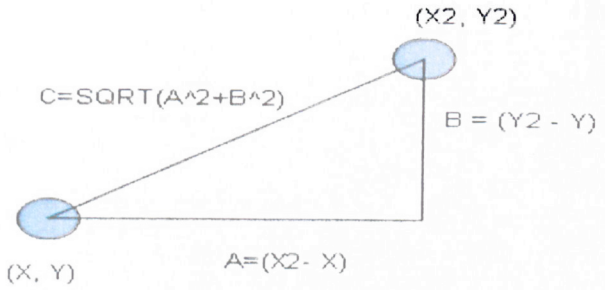

Figure 16.Pythagoreans Theorem

The nodes use a von Neumann Neighborhood to look around the deployment grid. To ensure that a point in the grid is within range, the distance formula is given in algorithm below:

distance(x, y, x2, y2, range)
if (x2 – x = 0 AND y2 -y =0) return false
else distance $\leftarrow \sqrt{(x2-x)^2-(y2-y)^2}$
if distance <= range return true
else return false

The distance formula uses the Pythagorean Theorem to determine the distance of two different nodes. The algorithm uses the x-y coordinates of the broadcasting node and a neighboring node to determine the sides of the triangle. The hypotenuse is then computed. If the hypotenuse does not exceed the nodes broadcasting range and the node is currently awake then the node is a valid neighbor. That node is then added to the list of neighbors of the broadcasting node. This process is repeated for all the wake nodes in the simulation.

3.5.3. Flooding Protocol: The data will disseminate from the sink and will travel across the network to all the different nodes as shown in the following algorithm:

```
Flooding(sender, receiver, visited)
        Enqueue Sender
        While Queue is NOT Empty
                current_node ← Dequeue Node
                if current_node = receiver
                        return true
                else
                        visisted.Add(current_node)
                        for each Neighbor in Node
                                if NOT visited.Contains(Neighbor)
                                        Enqueue Neighbor
```

When a node receives a packet of data using the flooding protocol it will retransmit it to all its neighbors. This process is repeated for all of the nodes that receive the message. Since only the intended receiver knows if the message is successfully delivered, the different nodes will continue to flood the network causing excessive packages from being transmitted. To avoid this and to provide the packets with a relatively short time-to-live a list of visited nodes is added to keep track of nodes that have already received the packets. By adding a list of visited nodes, any infinite loop within the network is prevented. Due to the highly dynamic structure of the resulting graph that the discovery algorithm produces, the flooding protocol is designed to be a Breadth First Search.

3.5.4. Power Production and Consumption: The main cause of power loss by a node is due to sending and receiving messages. Loss of power due to the amount of messages being sent can be simulated using a formula in Equation 1, derived from the nodes broadcasting range and the loss of energy per message sent.

$$\text{Energy} = \text{Energy} - \frac{\text{BroadcastingRange} * \text{Energy\_Per\_Msg}}{100} \qquad (1)$$

Energy loss is greater when sending a message, however just receiving a message causes energy to be used. Equation 2 demonstrates a possible formula to use in the loss of energy due to a message being received. In the equation, only a small

fraction of the amount of the energy per message is being lost. Other forms of energy loss come in the form of computation, and node overhead. To account for this type of loss, a new equation is used. Equation 3 shows how much energy is being lost due to overhead per time frame.

$$Energy = Energy - \frac{Energy\_Per\_Msg}{100} \quad (2)$$

$$Energy = Energy - Energy\_Per\_Step \quad (3)$$

Whenever the energy level of a node drops beyond a certain threshold, the node will enter a sleep mode. The node will remain in the sleep mode till it regains enough energy to become functional again. Energy in a node is produced primarily through the use of solar panels. The amount of energy that is being produced is limited and is dependent on the node's environment. By using an average node energy production formula in equation 4, one can estimate the amount of energy produced at every time frame.

$$Energy = Energy + Energy\_Produced \quad (4)$$

At every time frame, the node will lose power from messages being sent and received as well as from node overhead. While the amount of energy that the node is producing might not be enough to permit the node to run constantly, appropriate power management does allow the node to prolong its life allowing the network to continue functioning.

3.5.5. Programming: The simulator is written in C# using Microsoft Visual Studio.NET 2010. The application is in two parts: one module for the wireless sensor network, and the other is the simulator that hosts the WSN objects. The classes that

comprise the WSN are WirelessSensor, iSensorRadius, iSensorDelay, aPackets, aConnections, ConnectionCurrent, iResidualEnergy, WirelessSensorConnection, sSender and sReceiver, Packet, WirelessSensorNetwork, VectorList and Vector. This simulator supports a gigantic networks 400x400 nodes. Simulating transmissions across all of these connections, and then drawing the whole network several times per second, requires a nontrivial deal of processing. The CPU of a typical 3.2GHz machine is not enough for this task with the message pump that handles window events; hence, cramming both functions into one thread results in a completely unresponsive simulator window. Instead, the simulation thread is started in a separate thread, which runs continuously until the user stops it (clicking Stop Button).

3.6. Wireless Sensor Network Simulation: This application is a simulation of the WSN. The network is deployed based on parameters: network size (number of nodes), communications distance, energy costs for transmitting and receiving packets, etc. The network can then be used to simulate the detection of vectors traveling across the sensor network field. In this simulation, when a vector trips the sensor of a network node, the node generates a data packet and sends it to a downstream network node. The packets are routed appropriately until they reach a sensor within the "uplink zone" (the right side of the map, designated with a striped pattern). Each node also simulates an energy store, which is depleted by sending receiving packets, and by detecting vectors. Since the nodes have finite energy, they will eventually drop out of the communications network due to power down, causing network failure.

The simulation consists of two stages: deploying the network and running simulations. Before deploying the network, the properties of the network should be set using the

configuration sliders. The network configuration properties are grouped into two categories.

3.6.1. Network Configuration: these factors determine the hardware properties of the network. The following variables can be configured, Network Size, Sensor Radius, Sensor Period, Sensor Cost, Transmission Radius, Transmitter Period, Transmit Cost, Receive Cost.

3.6.2. Routing Parameters: These factors determine the software properties of the network: essentially, the packet-routing method to be used. If routing is set to "Random," each node selects a downstream connection randomly for each packet. If set to "Directed," the network routes packets based on the algorithm described in the Chang and Taissulas's work(Tassiulas, 1999).

3.7 Energy-Efficient Coverage

This research first develops some combinational results which are used to formulate the energy-efficient coverage problem as a linear integer problem.

3.7.1 Notations and definitions

Consider a network of N sensors intended to provide sensing coverage to a set of m target points in the region. The target points and the sensors are randomly deployed in the region. Sensors are deployed abundantly to provide maximum coverage lifetime and may have independent sensing radii. Let S be the set of all sensors and M be the set of all target points.

Definition 1: Utility of a sensor $s \in S$, $u(S)$, is the total number of target points $t \in M$ which lie in the sensing range of s.

Definition 2: Coverage of a target point $t \in M$ with respect to a set S of sensors, $C_S(t)$, is the total number of sensors $s \in S$ which cover the target point t.

Definition 3: A Feasible set S_f is a set of sensors s ϵ S such that for all target points t ∈ M, there is a sensor s∈ S_f which covers t.

Definition 4: A Cover Set A is a set of sensors s ϵ S such that,

- A is a feasible set.
- $\forall s \in A$: the $set\ A - \{s\}$ is not a feasible set.

Hence a cover set does not contain any abundant sensors and activating one such cover set is adequate to provide coverage in the network.

Definition 5: Utility of a cover set A is defined as

$$U(A) = \sum_{s \in A} u(s) \quad (1)$$

Definition 6: Coverage of the set M of target points with respect to a set S of sensors is defined as

$$C(M) = \sum_{t \in M} cx(t) \quad (2)$$

3.7.2 Properties

Next, some results related to the utility and coverage provided by a cover set are developed.

Lemma 1: For a cover set *A*, $C_A(M) = U(A)$.

Proof: Consider a target point $t \in M$. let X denote the set of sensors which cover *t* such that X ⊂ A. i have $c_A(t) = |X|$. Now for all $s \in X$, *t* adds one to the utility of sensor *s*, *u(s)* thus, *t* adds $|X|$ $(= c_A(t))$ to the utility of the cover set A. Also t adds $c_A(t)$ $(=|X|)$ to the coverage of the set M. Summing over all target points t ∈ M, i have the desired result.

Note that lemma 1 is also valid for any feasible set.

Lemma 2: For a cover set *A* of size *k (i.e.* $|A| = k$*)*, the utility of any sensor $s \in A$ is upper bounded by *(m+1-k)*.

Proof: Since A is a non-redundant feasible set, for all sensors s ∈ A there must be a target point t which is not covered any other sensor $s' \in A$ (otherwise the set A - $\{s\}$is a cover set and hence the set A is not a cover set) thus, i have

$$\forall s \in A, \exists t \in M : s \; covers \; t \text{ and } c_A(t) = 1.$$

Since $|A| = k$, there are at least k target points in M which have their coverage, with respect to the set A, equal to one, Now let $s_{max} \in A$ be the sensor node with maximum utility in A. since the total number of target points is m and there must be at least (k-1) target points which are not covered by s_{max}, $u_{(max)} \leq (m+1-k)$.

From Lemma 2 and the fact that $(|A| = k)$, i have the following corollary.

Corollary 1: the utility of a cover set A of size k is upper bounded as, $U(A) \leq (m+1k)k$.

Next i generalize the bounds to the utility of a cover set of any size.

Lemma 3: The utility of a cover set A is bounded as, $m \leq U(A) \leq (\frac{m+1}{2})^2, \forall m > 0$.

Proof: Since A is a cover set, each target point $t \in M$ must have coverage $c_A(t) \geq 1$. This together with the fact that $|M| = m$, *leads to* $C_A(M) \geq m$. Now fromLemma 1, the lower band follows.

From Corollary 1, i have $U(A) \leq (m+1-k)k$, where k = |A|. Let $f(k) = (m+1-k)\,k$. then from simple calculus, f (k) is maximized at k $= \frac{m+1}{2}$. Hence, U (A) $\leq (\frac{m+1}{2})^2$.

From lemmas 1 and 3, i observe that the coverage provided to the set of target points by any cover set is bounded only in terms of the total number of target points and these bounds are independent of the size of the cover set.

3.7.3 Linear Program

Next i address the issue of energy-efficient coverage in sensor networks. The objective is to find a feasible set *A (A⊂ S),* which comprises of a minimum number of sensors. It is easy to observe that the set A must be a cover set, as defined in Section *II-A.*

Definition 7: Excess Utility of a sensor $s \in A$, $u'(s)$, is defined as, $u'(s) = u(s) - 1$. Similarly, the excess utility of a cover set A is defined as $U'(A) = \sum_{s \in A} u'(s)$. Note that $U'(A) = U(A) - |A|$

Let S_E denote a set of sensors which provides coverage to the network utilizing minimum number of sensors. Then S_E is a cover set with minimum cardinality. That is, $\forall S' \subset S$, such that S' is a cover set, $|SE| \leq |S'|$. Since set S_E is a cover set, it also classifies the following constraints:

$$\forall_S \in S_E : u'(s) \geq 0 \quad (3)$$

$$\forall_t \in M : cs_E(t) \geq 1 \quad (4)$$

$$C_{SE}(M) = |S_E| + U'(S_E) \quad (5)$$

(3) is satisfied since otherwise the set $S_E - \{s\}$ would be a feasible set, and hence the set S_E would not be a cover set. (4) holds, since the set S_E is a feasible set. (5) follows from lemma 1. From (5), i have, $|SE| = C_{SE}(M) - U'(S_E)$. In other words, the set S_E minimizes over all sets of sensors *A*, the objective function *Y (A)* defined as,

$$Y(A) = C_A(M) - U'(A) \quad (6)$$

Subject to the following constraints

$$\forall_S \in A : u'(s) \geq 0 \quad (7)$$

$$\forall_t \in M : c_A(t) \geq 1 \quad (8)$$

To pose the above optimization problem as a linear program, let us introduce the following variables. For each sensor node, si, where $i \in [1 \ldots . N]$, let xi denote whether $s_i \in A$. That is, $\forall_I : 1 \leq i \leq N$, the variable x_i takes on value

- 0, if $s_i \notin A$.
- 1, if $s_i \in A$.

For each target point $t_j \in M$, where $j \in [1..m]$, let y_{ij} denote whether the target point t_j is covered by sensor s_i. Precisely, $\forall_i \in [1..N]$, $\forall_j \in [1..m]$, the variable y_{ij} takes on value

- 1, if sensor s_i covers target point t_j.
- 0, otherwise.

Note that the variables y_{ij} could be precomputed once the network configuration is known, and are effectively constant thereafter. Now the energy-efficient coverage problem can be expressed as the following linear integer program H. Christos, Papadimitriou and Keneth S.(1998) with N variables and N + m constraints.

Minimize $\sum_{i=1}^{N} x_i$ subject to

$$x^i(\sum_{j=1}^{m} y_{ij} \quad \forall_i \in [1..N] \quad (9)$$

$$\sum_{i=1}^{N} x_i y_{ij} \geq 1 \ , \forall_j \in [1..m] \quad (10)$$

(9) and (10) correspond to (7) and (8) respectively.

From the structure of the above optimization problem, i observe that minimizing the objective function Y(A) (Equation 6), is the same as minimizing the sum of the coverage of all target points with respect to A and maximizing the sum of utilities of all sensors s ∈ A, simultaneously.

3.8 **Lifetime Coverage and Connectivity Problem**

This problem differs from the energy-efficient coverage problem posed in section II in the following aspects:

- Each set of sensors needs to ensure both coverage and connectivity in the network.
- The aim is to find multiple such disjoint (or mutually exclusive) sets, and in particular the maximum of such sets.

In this section the problem is formulated more precisely and there will be comments on its optimal solution.

3.8.1 **Problem Formulation**

Consider a region of interest represented by a Manhattan grid, with m target points distributed randomly in the grid Cardei. M and J. Wu.(2004). N static sensor nodes are deployed randomly in the region. Sensors sense information and transmit it to the monitoring station, which is located at the center of the grid. Every sensor has a sensing capability covering a disk of radius R_c. A sensor s whose distance from another sensor S' (or the monitoring station) is less than its transmission radius (R_c), can transmit directly to s' (or the monitoring station). All the sensors have a constant charge level C at the time of deploying.

An active sensor has the ability to sense as well as transmit information to its surrounding sensors, which lie within its transmission radius (R_c). All active sensors can also act as relay nodes. A Passive sensor is unable to sense or relay information.

An available sensor is one which has a nonzero energy level and is currently inactive or passive.

Let S be the set of all sensors and M be the set of all target points. Let A_e denote the set of available sensors (which is initially the same as S).

Definition 8: An active sensor s has a path to monitoring station MS in the set S iff s can transmit to MS or there are relay sensors $r_1, \ldots\ldots, r_k \in S$ such that

- Sensor s can transmit to sensor r_1.
- $\forall_i : 1 \leq i < k$, sensor r_i can transmit to sensor r_{i+1}
- Sensor r_k can transit to MS.

Definition 9: An active set, A, is the set of sensors $s \in S$ such that

- A is a feasible set.
- $\forall_s \in A$: there is a path from s to MS in A.

An active set is adequate to maintain coverage and connectivity requirements in the network for a while. Let such a set of sensors, A, be activated at time T_1. At some time $T_2 > T1$, one or more sensors in the set A would get consumed of their energy completely. At this time a new set of sensors $s \in A_e$ needs to be computed, which shall be activated to satisfy the above requirements.

Assuming all the sensors in the set A die at the time, the new set should be mutually exclusive to all the active sets computed earlier. Thus the computed active sets could be activated in succession, where each one of them contributes towards one round of sensor nodes activation in the network.

Given a particular network configuration, let R be the number of rounds thus computed (i.e. R = number of disjoint active sets computed fro S). Assuming the

sensors spend trivial amount of energy in the sleep or inactive state, the number of rounds, R, represents the network lifetime. That is, if duration of a round is T, the network lifetime equals RT. Hence, a large computed R would imply longer network lifetime. On the contrary, a trivial activation decision could be set the initial active set A to be the same as S, which results in minimum R(= 1) or minimum network lifetime.

The objective of the lifetime coverage and connectivity problem is to compute the maximum R and to compute the active sets (A_1, ..., A_R) corresponding to the round (1, ..., R), or equivalently,

Maximize R such that $\forall$ i, j $\in$ [1, ..., R] :

- A_i is an active set.
- $A_i \cap A_j = \emptyset$, $I \neq j$.

Through multiple solutions, the one with least energy consumption is desired. For example, consider two solutions have the same value of R but different set of disjoint active sets (A_1, ..., A_R) and (A'_1, ... A'_R) respectively. The solution consists of those active sets for which the total number of sensors activated is the least (i.e. the former set is a solution if $\sum_{i=1}^{R} |Ai| \leq \sum_{i=1}^{R} |A'i|$).

3.8.2 NP Completeness of the Optimal solution

Finding the maximum number of disjoint active sets of sensors in S, where each of the active sets can independently provide coverage and connectivity in the network is the optimal solution to the lifetime coverage and connectivity problem.

Cardei and Du have shown that computing an optimal set cover (or the lifetime coverage problem) is NP-complete. The problem that is faced also needs to compute

an optimal set cover. Moreover, the computed sets must also guarantee connectivity in the network. Hence, this problem has a solution represented in the following section.

3.9 **Results and Discussion**

The initial results of the simulator provide promising results. There are seven (7) parameters in the experiment. The network size parameters will the main comparison standard in the conclusion. The comparison will be done using network size as the main parameter and it will be compared with the other six (6) parameters. For each comparison the best trend of comparison will be selected .If the number of network size start from15up to 45, then i will almost a linear increment in the transmitter delay. And if the number of network size start from30 up to 45, then i will almost a linear increment in the sensor cost. And if the number of network size start from35up to 55, then i will almost a linear increment in the transmission cost. The GUI is able to provide the user with the visualization of the WSN and immediate changes in the different constraints are displayed. The main constraints that the visualization will be looking at are how the nodes are deployed and the appropriate size of the network. The neighbor discovery process and the impact of the number of neighbors have on the network. Furthermore, the protocol being used and the effects that it has on the node's power supply are critical to the overall life of the network. The following tables and figures are showing and demonstrating the simulation results over two different rounds and set of parameters. In the first attempt of the simulation, that set the value of network size to 10 nodes, which is relatively a small number. The reason for this choice is purely experimental (inspecting the reaction of the developed algorithm in the simulation). The initial energy is set to 1000 Joules, which will be optimized during the course of the simulation. The sensor radius is set to 45 with a maximum

noted movement in X and Y directions of 515. It can be clearly seen that the resulted sensor delay is 15, transmitter delay 10, transmission cost 2, receiver cost 15 and sensor cost 20. These values are yet to be proven whether they represent an improvement or not. Thus, running the second round with different parameters values to prove the enhancement in the performance is WSN.

Sensor nodes are energy-constrained and hence only a minimal set of sensor nodes needs to be activated at any given time. Node activation schedules ought to be in succession in order to fulfill maximum lifetime of the network. Eventually, the optimal solution to the lifetime coverage and connectivity problem is NP-Complete. It is possible to compute the optimal solution by extensive search for small network sizes (small values of N and m). However, the complexity of computing the optimal solution grows very rapidly with the size of the network. Hence, an efficient heuristic solution approach requires to be developed.