2018

Wireless Sensor Network Clustering with Machine Learning

Larry Townsend
Nova Southeastern University, larryt4@gmail.com

Wireless Sensor Network Clustering with Machine Learning

by

Larry Townsend

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science

College of Engineering and Computing
Nova Southeastern University

2018

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Wireless Sensor Network Clustering with Machine Learning

by
Larry Townsend
August 2018

Wireless sensor networks (WSNs) are useful in situations where a low-cost network needs to be set up quickly and no fixed network infrastructure exists. Typical applications are for military exercises and emergency rescue operations. Due to the nature of a wireless network, there is no fixed routing or intrusion detection and these tasks must be done by the individual network nodes. The nodes of a WSN are mobile devices and rely on battery power to function. Due the limited power resources available to the devices and the tasks each node must perform, methods to decrease the overall power consumption of WSN nodes are an active research area.

This research investigated using genetic algorithms and graph algorithms to determine a clustering arrangement of wireless nodes that would reduce WSN power consumption and thereby prolong the lifetime of the network. The WSN nodes were partitioned into clusters and a node elected from each cluster to act as a cluster head. The cluster head managed routing tasks for the cluster, thereby reducing the overall WSN power usage. The clustering configuration was determined via genetic algorithm and graph algorithms.

The fitness function for the genetic algorithm was based on the energy used by the nodes. It was found that the genetic algorithm was able to cluster the nodes in a near-optimal configuration for energy efficiency. Chromosome repair was also developed and implemented. Two different repair methods were found to be successful in producing near-optimal solutions and reducing the time to reach the solution versus a standard genetic algorithm. It was also found the repair methods were able to implement gateway nodes and energy balance to further reduce network energy consumption.

We hereby certify that this dissertation, submitted by Larry Townsend, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Wei Li — 8/15/2018

Wei Li, Ph.D.
Chairperson of Dissertation Committee

Date

8/15/2018

Michael J. Laszlo, Ph.D.
Dissertation Committee Member

Date

8/15/2018

Sumitra Mukherjee, Ph.D.
Dissertation Committee Member

Date

Approved:

8/15/2018

Meline Kevorkian, Ed.D.
Interim Dean, College of Engineering and Computing

Date

College of Engineering and Computing
Nova Southeastern University

2018

Table of Contents

Abstract ii

Chapters

List of Figures

List of Tables

Introduction

Background

Wireless sensors are small devices that make local measurements such as environmental conditions like temperature or pressure and contain the hardware necessary to transmit this information to other devices. They may be dropped or scattered over an area to provide information on current conditions. The sensors are typically reliant on internal battery power and therefore have a limited lifetime (Abbasi & Younis, 2007). The sensors may form a network known as a wireless sensor network (WSN) in order to communicate with each other and also send information to a special node on the network called a base-station or sink. The sink transmits information to other systems where it is used for analysis (Akyildiz, Su, Sankarasubramaniam & Cayirci, 2002). WSNs have no pre-determined infrastructure and therefore the network must be set up in an ad hoc manner. WSN nodes may have limited mobility such as sensors floating in the ocean but generally are in fixed positions. Unlike traditional wired networks which typically have dedicated hardware to route network traffic, each node in a WSN acts as a router. Each WSN node either communicates directly with other nodes within its transmission range or uses other nodes to relay messages to nodes outside its range (Zhou & Haas, 1999). Several typical WSN architectures are shown in Figures 1, 2, and 3 below. This architecture allows WSNs to be set up very quickly and with "relatively low cost" (Zhou & Haas, 1999). There are many situations where WSNs are useful including military applications, emergency response, or natural disasters (Abbasi & Younis, 2007).

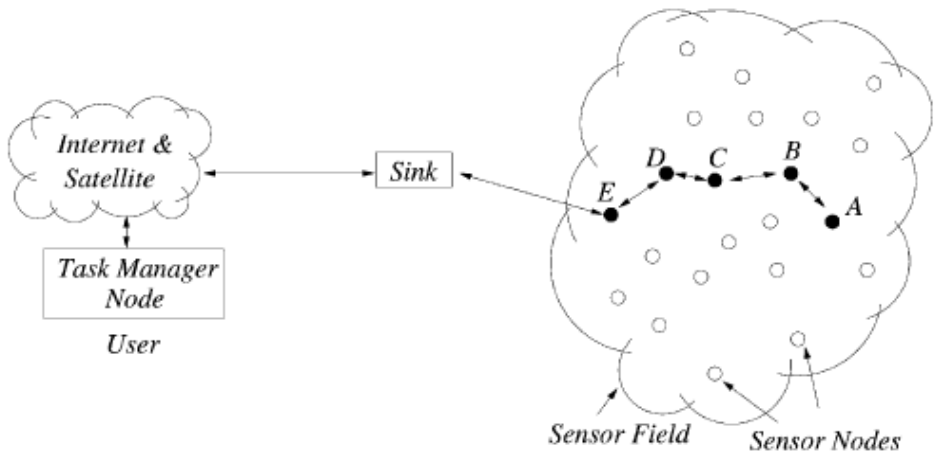

Figure 1. WSN (Akyildiz et al., 2002)

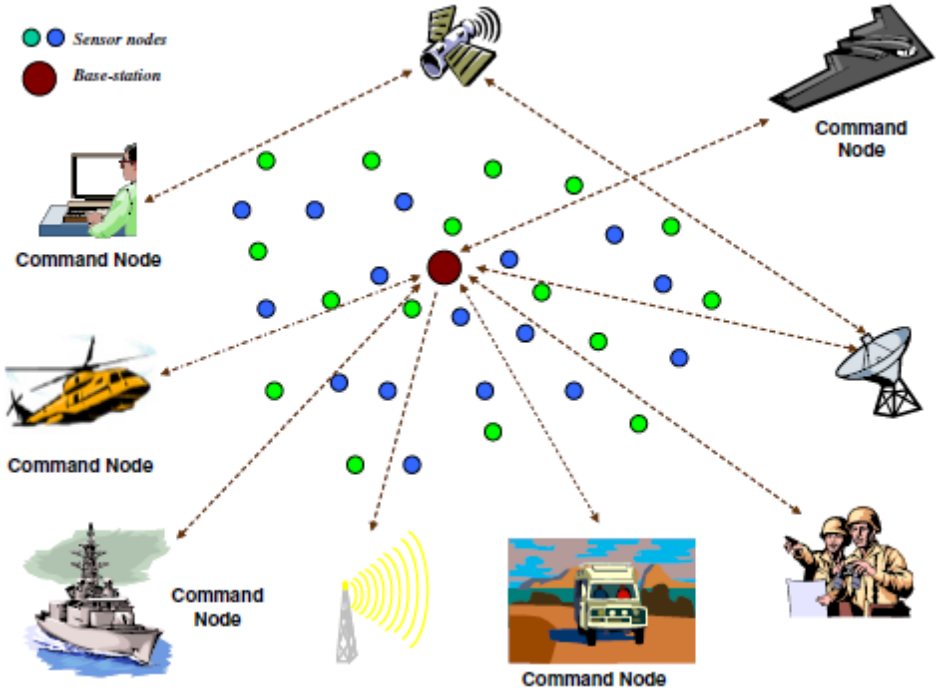

Figure 2. WSN (Abbasi & Younis, 2007)

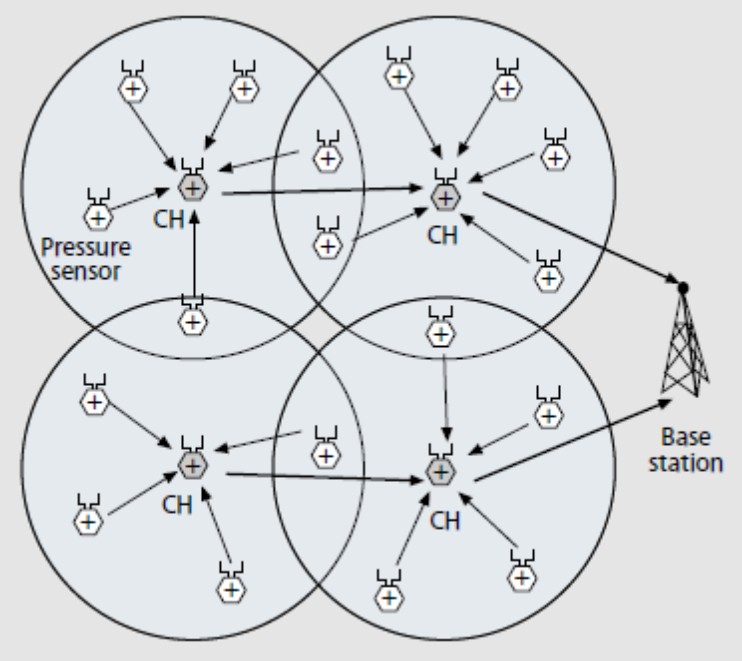

Figure 3. WSN (Younis, Krunz & Ramasubramani)

These same features that make WSNs attractive for the uses noted above also create additional resource and security issues as compared to standard wired and wireless networks (Blum et al., 2004). With no central authority or management, each node in a WSN, along with acting as a router, (Safa, Artail & Tabet, 2010) must also run its own intrusion detection (Mohammed, Otrok, Wang, Debbabi & Bhattacharya, 2008). An additional challenge with WSNs is the limited resources available. The mobile devices used as nodes of a WSN typically have limited battery power and therefore any protocols devised for WSNs should strive to limit increases in network overhead and CPU load (Abbasi & Younis, 2007).

WSN nodes have to perform additional work as compared to traditional wired network nodes. WSN devices are battery powered and therefore power management is an issue. One of the methods that has been proposed to address the issue of limited power is to use clustering. Through grouping nodes into clusters, a node can be elected to manage routing and intrusion

detection for each cluster. In this manner, overall resource usage of the network can be reduced (Chinara & Rath, 2009).

There are many attempts in the literature to improve WSN and MANET resource consumption by implementing a clustering algorithm, either separately (Cheng, 2012; Abbasi & Younis, 2007; Mohammed, Otrok, Wang, Debbabi & Bhattacharya, 2011) or as part of a routing protocol (Hajami, Oudidi & ElKoutbi, 2010; Safa et al., 2010). The idea behind these efforts is that groups of nodes can be clustered together and a cluster head can be chosen to act on behalf of the cluster. The cluster head handles routing updates and intrusion detection for its local cluster (Zhang et al., 2009). Therefore, the use of the cluster head reduces the resource load on the cluster member nodes. There are several proposed methods to elect cluster heads in the literature, including random selection, connectivity-based selection, or selection based on remaining resources (Mohammed et al., 2011).

Younis & Fahmy (2004) presented a case study showing specifically that the lifetime of a WSN can be prolonged using clustering. In this context, the lifetime of a network is defined as how long the network is in operation until the first node has its energy depleted to where it can no longer function as part of the network. In this context, a set of cluster heads was elected and then nodes nearby were grouped with a cluster head. The cluster head then assumed responsibility of tasks for its cluster members. Tasks coordinated by the cluster head included communication, both internal to the cluster and external, as well as data aggregation. Younis & Fahmy (2004) also note that clustering can reduce overall network energy usage since it has been shown to reduce network communication overhead and also prolong network lifetime through re-clustering with the intent to rotate cluster heads to nodes with higher residual energy. Clustering was also noted to increase network lifetime through “reducing the number of nodes contending

for channel access" (Younis & Fahmy, 2004) and "routing through an overlay among cluster heads, which has a relatively small network diameter" (Younis & Fahmy, 2004).

The authors implemented a clustering scheme called *HEED* that was based on the residual energy of the nodes along with a "secondary parameter such as proximity to its neighbors or node degree" (Younis & Fahmy, 2004). The clusters were configured such that each node was a member of exactly one cluster and cluster heads were located such that all nodes were within transmission range of at least one cluster head. *HEED* (Younis & Fahmy, 2004) was compared to an existing algorithm from the literature known as *LEACH* (Heinzelman, Chandrakasan & Balakrishnan, 2002). *LEACH* had been shown to improve network lifetime significantly over static network clustering (Heinzelman, Chandrakasan & Balakrishnan, 2002). It was found that the network utilizing HEED was able to exist for approximately double the length of time as LEACH until the first node had died. This clearly demonstrates that significant network energy savings can be achieved through the use clustering.

The case study of Younis & Fahmy (2004) detailed above was implemented as part of a sensor network where there was no node mobility within the network. A WSN or a MANET where there is little or no node mobility can be considered an undirected graph. The nodes of the network correspond to vertices of the graph (Rajan, Chandra, Reddy & Hiremath, 2008). Therefore, the problem of clustering WSN nodes can be viewed as a graph-partitioning problem. As shown below, given that the problem can be viewed as graph partitioning and it involves selecting a number of points to be the optimal locations for cluster heads, the problem can be viewed as a larger class of problems known as location problems (Reese, 2006). Particularly the class of location problems known as p-median problems is applicable as an analog for clustering the nodes of a WSN.

Location problems typically involve placement of new facilities. The problem is to place the facilities such that the cost of access or distance to the facilities by other members of the set of objects is minimized (Mladenović et. Al., 2007). This has been shown to be an NP-hard problem even in simple configurations (Kariv & Hakimi, 1979).

The p-median problem is a location problem and was described in Laporte et al., (2015) and Hakimi (1964). The p-median problem space can be considered an undirected graph *G(V,E)* as was shown to also be the case with wireless sensor networks (WSNs) (Younis & Fahmy, 2004). Each vertex *v* is assigned a weight *w(v)* and each edge a length *l(e)*. The distance between a vertex *v* and a set of points X_p on *G* is defined below. The points X_p are located "along any edge of G and may or may not be a vertex of G" (Kariv & Hakimi, 1979).

$d(v, X_p) = min_{1 \leq i \leq p}\{d(v, x_i)\}$ (Kariv & Hakimi, 1979)

The distance-sum is defined as:

$H(X_p) = \sum_{v \in V} w(v) \cdot d(v, X_p)$ (Kariv & Hakimi, 1979)

The p-median is defined as the set of points X_p such that the distance-sum is minimized. Kariv & Hakimi (1979) showed that there exists a p-median where the set of points are vertices; $V_p = X_p$. This means that although the set of points X_p that minimize the distance-sum could exist anywhere on the graph *G*, there exists a set of points V_p that are vertices of the graph that also minimize the distance-sum. The p-median problem is the task of discovering the set of points that make up the p-median for a given graph *G*:

$H(V_p) = min_{X_p \; on \; G}\{H(X_p)\}$ (Kariv & Hakimi, 1979)

The equation above assumes $p < |V|$ and that *G* does not contain loops or multiple edges (Kariv & Hakimi, 1979).

As noted above, the specific constraints that shall be applied to the undirected graph will be that every vertex v must either be a member of V_p or within one distance unit (1-radius) of a vertex in V_p and there will be a minimum threshold that number of points $|V_p|$ must be above to ensure full network connectivity. Above this threshold, the number of points that make up the p-median may be varied in order to minimize the target function. These constraints will make the solution applicable to wireless sensor networks with little to no node mobility.

A proof provided in Kariv & Hakimi (1979) showed that the p-median problem is NP-hard, even in a situation with a very simple, planar graph. In the proof the length of all edges was set to one and the cost of all vertices also set to one. The problem addressed in this dissertation can be reduced to the p-median problem described in the proof in Kariv & Hakimi (1979). Applying the 1-radius constraint effectively sets the length of all edges to one. Assuming all the nodes to be identical sets the cost of vertices to the same value. Additionally, restricting the value of p to a constant (above the required threshold) and assuming $p < n$, results in the same complexity as in the proof presented in Kariv & Hakimi (1979). Therefore the problem addressed in this work is NP-hard.

As shown, the problem of clustering a WSN can be viewed as the p-median problem with specific constraints. In WSNs the costs of connections are based on the number of network hops instead of Euclidean distance because the power consumed in transmitting a packet is assumed to be the same for all nodes when the destination is within the transmission range of the sending node. Also, the radius from the facility to other elements is not typically constrained in the p-median problem. Therefore, rather than minimizing the distance-sum as in the formal definition of the p-median problem, the target function will be based on minimizing the total network energy usage. Although using different measurements, the task presented in this dissertation is

similar to the p-median problem, identifying a set of nodes to minimize cost-based target function and the methods proposed in this work may be extended to the p-median problem.

Problem Statement

The intent of this work was to use specifically genetic algorithms and graph algorithms to solve the p-median problem constrained in a manner to make it applicable to wireless sensor networks (WSNs). The solution took the form of a graph partition into two subgraphs, one that formed the primary communications path for the network and a second subgraph where the member nodes were connected to the nodes of the first set. In order to be considered feasible these partitions have to provide a communication path for all nodes of the network to a sink node. Successful implementation of this solution resulted in a more energy efficient network and thereby increased the lifetime of the network.

We can consider the nodes of a WSN as vertices on an undirected graph *G(V.E)*. In addition to the sensor nodes there are sink nodes *S* that are not members of *V* but are part of the partition problem. Sink nodes are not considered in the measurement of energy usage since sinks are typically connected to the grid or a more substantial power source and are not limited to internal battery power as with the sensor nodes. Unlike traditional undirected graphs, within a WSN the edges are defined based on which nodes are within transmission range *R* of each other. This is expressed in the function $MA_{Connect}$ shown below, where $d(v_i, v_j)$ represents the distance from v_i to v_j.

$$MA_{connect}(v_i, v_j) = \begin{cases} 1\ ;\ d(v_i, v_j) \leq R \\ 0\ ;\ d(v_i, v_j) > R \end{cases} \tag{1}$$

Using $MA_{Connect}$ the graph of the WSN *W* can be defined as:

$W = (V \cup S, E)\ where\ E = \{(v, w) | MA_{Connect}(v, w) = 1\}$

In *W* the number of nodes and positions are fixed. In order to achieve the goal of this work *W* is partitioned into two sets of nodes, cluster heads (*CH*) and member nodes (*B*). Using these sets of nodes two subgraphs of *W* are defined; *BLUE* and *RED* as shown below.

$BLUE = (CH \cup S, E_{BLUE})\ where\ E_{BLUE} \subset E$

$RED = (B \cup CH, E_{RED})\ where\ E_{RED} \subset E\ and\ \forall\ (v, w) \in E_{RED}: v \in B\ and\ w \in CH$

CH nodes receive messages from *B* nodes that are connected to them via the *RED* graph. Messages are transmitted from *CH* nodes to the sink(s) via the *BLUE* graph.

In order to be considered feasible the network graph *W* has to provide for the communication of all nodes to a sink. In order for this to be true both the *BLUE* and *RED* partitions have to be feasible. The *BLUE* graph is feasible if and only if all *CHs* are connected through the *BLUE* graph to a sink node. The RED graph is feasible if and only if all member nodes(*B*) are connected to a cluster head(*CH*).

Using the above definitions there are potentially many feasible partitions of *W* into *RED* and *BLUE*. Different partitions result in different energy consumption rates and therefore a longer or shorter network lifetime. The target function *TF*(), detailed in the formal definition below, is used to calculate the energy consumed by the network over a fixed period of time. The optimal partition of *W* into *RED* and *BLUE* minimizes the target function and thereby results in the longest network lifetime. A formal definition of the problem follows below.

Formal Definition

Given a set of nodes $V = \{v_i \mid 1 \leq i \leq n\}$ in an undirected graph *G* and an integer *m*, such that $0<m<n$, the problem is to partition V into *m* clusters ($C_1, C_2, \ldots, C_m$) and to identify cluster heads $CH = \{ch_i | 1 \leq i \leq m, ch_i \in V\}$. The resulting network configuration shall be such that each cluster has exactly one cluster head, there exists a path from all nodes to a sink,

and the target function *TF*() is minimized. The set of sink nodes *S* is defined as $S = \{s_i | 1 \leq i \leq c, s_i \notin V\}$ where *c* indicates the number of sinks.

Distance within the wireless sensor network is defined as three-dimensional Euclidean distance. The distance between two nodes $q \in V$ and $p \in V$ is therefore given by:

$$d(q,p) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2 + (q_z - p_z)^2} \quad (2)$$

Using the distance function, $MA_{connect}$, the function that identifies if nodes are connected by an edge is defined below.

$$MA_{connect}(v_i, v_j) = \begin{cases} 1\ ;\ d(v_i, v_j) \leq R \\ 0\ ;\ d(v_i, v_j) > R \end{cases} \quad (1)$$

The graph partition can be converted into partition of two subgraphs, *BLUE* and *RED*. The *BLUE* graph represents the cluster heads (*CH*s), connections between *CH*s, and connections from *CH*s to sinks. The *RED* graph represents the member nodes $B = \{b_i | 1 \leq i \leq n, b_i \in V, b_i \notin CH\}$ and connections from member nodes to *CH*s. The *RED* graph is feasible if and only if all of the nodes in the *RED* graph are connected to a *CH*. Using $MA_{connect}$ above, a Boolean function RED() can be defined as:

$$RED(B) = \begin{cases} true\ ;\ if\ \forall\, b \in B: \sum_{i=1}^{m} MA_{connect}(CH_i, b) \geq 1 \\ false\ ;\ otherwise \end{cases} \quad (3)$$

The above equation indicates that the *RED* graph is feasible if and only if all member nodes are within transmission range of at least one cluster head. The *BLUE* graph is feasible if and only if all *CH*s are connected through the *BLUE* graph to a sink node $s \in S$ as indicated in equation (4) below. In equation (4), S_{comp} represents the set of nodes that form the component of the sink(s).

$$BLUE(CH) = \begin{cases} true \ ; \ \ if \ \forall \ ch \in CH: ch \in S_{comp} \\ false \ ; \ \ otherwise \end{cases} \tag{4}$$

Initially in this work network configurations will only be considered feasible if all sinks are connected to the same component. Sinks with separate components are considered separate networks. It is feasible that there exist network topologies such that separate sink components facilitate less overall energy usage and additional work to explore this may be done in the future.

The function that defined the energy consumption of a wireless sensor network over a given time interval is expressed as $f(E_{recv}, CH)$, where E_{recv} is the energy consumed by a node to receive a message. The function is defined below for a finite time period that is a small fraction (<1%) of the typical network lifetime.

$$f(E_{recv}, CH) = \sum_{i=1}^{n} [x_i * E_{send} + y_i * E_{recv}] + e|CH|$$

$$= \sum_{i=1}^{n} [x_i * k + y_i] * E_{recv} + e|CH| \tag{5}$$

In equation (5) the variables x_i and y_i represent the number of messages being sent and received at v_i respectively. The constant *k* is a factor representing the extra energy required to send versus receive transmissions as defined in (Wu et al., 2002).

$E_{send} = kE_{recv}$, $k \geq 1$ (Wu et al., 2002)

The last variable *e* accounts for the extra energy consumed by cluster heads versus member nodes. It represents the energy required for the average cluster head to maintain routing information and actively listen for messages from member nodes. Sinks are typically connected

to a substantial power source and are therefore not considered in the problem to minimize the energy usage of the network.

Given the above, the target function *TF*() to be minimized in this work can now be defined for a given network configuration as shown in equation (6) below.

$$TF() = \begin{cases} f(E_{recv}, CH) \; ; & if\ RED(B) = true\ AND\ BLUE(CH) = true \\ \infty & ;\ otherwise \end{cases} \tag{6}$$

As defined in equation (6), the energy definition function $f(E_{recv}, CH)$, is only valid when the network is feasible, meaning that all nodes are able to communicate through a path with at least one sink. A network configuration is considered feasible when the *RED* graph is feasible, and the *BLUE* graph is feasible. When a network configuration is not feasible, meaning either the *RED* or *BLUE* graph are not feasible, then the configuration receives infinity as the score. Since this is a minimization problem, infinity indicates the poorest performance for a configuration.

The problem as presented above shows that the p-median problem can be constrained in a manner to emulate a WSN and therefore the work presented here on improving the lifetime of a WSN may be extended to be applicable to a greater class of problems.

Dissertation Goal

As the literature review below shows, many clustering algorithms for WSNs were successful in reducing network overhead or improving resource utilization as compared to the existing protocols. However, the success occurs typically only within a certain range of environmental factors such as a mostly static network (Wu, Cao & Raynal, 2009), greater than a certain number of nodes (Chauhan, Awasthi, Chand & Chugh, 2011), or within a certain transmission range (Zhang, Ng & Low, 2009).

It is also shown in the literature review that there are several common characteristics that are desirable for a WSN clustering algorithm:

- It should improve network scalability by reducing network overhead associated with routing messages (Er & Seah, 2010).
- It should reduce energy consumption via lower computational overhead, thereby improving the lifespan of nodes with limited battery power (Chauhan et al., 2011).
- It should improve the successful delivery of packets (Safa et al., 2010).
- It should not significantly increase overall network resource usage due to cluster maintenance (Zhang et al., 2009).

The goal of this research was to develop a clustering method to prolong the lifetime of a WSN. This meant reducing the overall resource usage of the network without negatively impacting the delivery of packets. These characteristics were satisfied over a wide range of environments. The measurement of success was through reducing power usage and prolonging the network lifetime. Improving scalability and packet delivery are beyond the scope of this work.

Research Questions

As can be seen in the literature there have been many attempts to improve clustering of WSNs. The research questions that arise from a review of the literature involve attempts to provide meaningful comparisons to this existing body of work and also improvements upon it. Therefore, the primary questions to be addressed by this research are as follows:

1. Which are the most useful metrics for measuring the quality of clustering for WSNs?

2. Is it possible to adapt a genetic algorithm to identify clusters in WSNs?
3. Is there value in considering a WSN a capacitated p-median problem?
4. Can a genetic algorithm optimize clustering of a WSN when the nodes have variable transmission range?

Relevance and Significance

As noted earlier there are many practical applications for WSNs including military and emergency or rescue situations (Abbasi & Younis, 2007). In both these situations maintaining communications is critical. Soldiers need to maintain communication to coordinate their movements and alert others to danger. Reducing the resource usage of WSN nodes would allow soldiers to stay in communication longer. Cluster head schemes would also provide the node management and communication required for improved intrusion detection (Zhang & Lee, 2000). Similarly, in rescue operations, allowing rescue workers to stay in communication longer would facilitate longer search and rescue operations. The proposed research benefits both these implementations as well as other situations where a low-cost network that can be set up quickly is required.

Much of the previous work related to clustering algorithms for WSNs has focused on particular performance aspects. Some focused on reducing routing overhead through a reduction in the number of routing messages. Other work focused on power usage, attempting to keep nodes active as long as possible by reducing CPU load to preserve battery life. There are also examples where cluster overhead was the focus as compared to other clustering algorithms or protocols. When compared to more commonly used protocols and clustering schemes in previous work, many of these more recent approaches were successful in improving a particular aspect of WSN performance. While all of these aspects are important, trade-offs were often

necessary to achieve a singular goal. This dissertation focused on power usage of WSNs in order to prolong the lifetime of the network. However, rather than focusing on one aspect of power usage, this work evaluated the most important factors in power consumption. Factors considered in power consumption included message delivery efficiency and cluster management overhead.

Additionally, the overall power usage of the network was considered using different clustering schemes. This comparison was required since different schemes resulted in different numbers of nodes being in an active versus passive mode and also changed the amount of network maintenance required at each individual node. The relative power usage of each factor was compared and the work focused on the factors determined to be significant in prolonging the life of the network through reduced power consumption. The significant result of this research was an algorithm that improved network energy wfficiancy across a range of environments.

It is likely that much of this work may be applicable in other situations. Any circumstance where clustering of data points is required could benefit from this work. This work will also be useful in the broader field of location problems such as the p-median problem.

Barriers and Issues

There are several factors that make developing an ideal clustering system for wireless sensor networks (WSNs) inherently difficult. As noted above, finding the optimal clustering solution is in general an NP-hard problem. Therefore, the clustering solution was difficult to measure since brute force methods even on a small p-median problem are not feasible. There are p-median problem data sets with known optimal solutions but these are not constrained as would make them applicable for WSNs. Also, any solution must be scalable, since WSNs may scale up to thousands of nodes depending on the application (Abbasi & Younis, 2007).

Along with the primary task of developing the clustering algorithm there are peripheral issues that complicated the proposed work. There is no standard set of benchmarks to test the implementations against. Therefore, a reasonable set of metrics had to be developed as part of the work.

Assumptions, Limitations and Delimitations

Assumptions. Within the context of this work it was assumed that the sensor nodes being modeled were in good working order and functioning per design. There existed no defects in the nodes that caused them to lose power prematurely or have their power drained due to reasons other than operations used in typical network operations such as sending and receiving messages and maintaining routing information as applicable. The nodes were also expected to be of the same type and manufacture so that they all had the same transmission range and consumed the same amount of power for the same operations. It was also assumed that wireless transmissions sent by nodes were received by nodes within range and were not blocked or corrupted due to environmental or other external conditions. While variable transmission distances and power levels may be modeled in future work, this assumption eliminated the need to account for resending a proportion of messages. Also it was assumed that the sink had sufficient computing power to perform the clustering calculations in a reasonable time.

The assumptions of no defects in the nodes and 100% successful message transmission may not be typical of what is encountered in real-world situations. However, these assumptions did not affect the results of the work being proposed. The intention was that these assumptions were held true for all network clustering schemes being compared and therefore they did not provide an advantage to one scheme over another. The assumption that all nodes were the same was not unreasonable given that the nodes of a network will typically be deployed to monitor the

same values; temperature, seismic, motion, etc., and therefore using the same type and manufacture of node would reduce variability among the data collected from different nodes. The last assumption related to computing power at the sink was not unreasonable since a standard PC should provide sufficient power for the calculations required.

Limitations. The primary limitation for this work was the lack of relevant solved data sets. Although there are solved data sets for the p-median problem where the optimal configuration is provided, they do not include the constraints given in the problem statement for this proposed work in order to make the solution applicable for WSNs. The intent of this work was to ultimately compare the results of various clustering schemes and therefore this limitation did not affect the validity of the work. However it should be known how close the proposed solution is to providing theoretical optimal solutions.

An additional limitation of the proposed work was that the feasibility of the solution will only be tested using software simulations of WSNs. A real-world test of the proposed solution was not feasible due to time and resource constraints. This did not reduce the validity of the work since this is typically the case for similar approaches in the literature and the comparisons of different approaches were performed using similar methodologies.

Delimitations. In order to make the p-median problem applicable to WSNs the networks were constrained to have cluster sizes no larger than a fixed-radius, meaning that the radius of any cluster could not be larger than the transmission range of the nodes. In order to facilitate communication between clusters the distance between cluster heads was also constrained to less than twice the transmission range. Also each cluster had to have exactly one cluster head and

each node had to belong to at least one cluster. Additionally this research only considered networks with one sink.

Summary

Wireless sensor networks (WSNs) have been shown to be useful in many situations where a network needs to be set up quickly and no fixed infrastructure exists. The individual sensor nodes of a WSN are typical small devices that rely upon battery power. Therefore there has been much work done toward the goal of reducing the power usage of the individual nodes and thereby prolonging the lifetime of the network. One method of reducing power usage of a WSN is to cluster the nodes and assign a cluster head for each cluster. A cluster head maintains routing information and combines messages from the nodes within its cluster. It has been shown that machine learning techniques are applicable to determining the optimal cluster configuration. The method that was focused on in this work was a genetic algorithm.

The problem of clustering the nodes of a WSN can be considered a constrained version of a larger class of problems known as location problems and more specifically the p-median problem. The p-median problem is the problem of locating facilities so that there is minimal cost of travel for all demand units. The cluster heads within a WSN can be considered the facilities with the remaining nodes of the WSN the demand points. In order to approach the node clustering problem as an optimization problem like the p-median problem, a target function was created that included terms for network architecture and location of cluster heads as well as the energy required for sending and receiving message and the number of messages sent. Constraints on the problem included limiting the radius of clusters to the transmission range of the nodes, requiring that each cluster has exactly one cluster head, and that each node belong to at least one cluster.

There are several difficulties in clustering the nodes of a WSN. Finding the optimal cluster configuration of a network is an NP-hard problem and a brute force approach will not work even with a small network. Also, there does not exist a set of solved problems to benchmark results against therefore the proposed solution had to be compared to other approaches.

Review of the Literature

The literature review is presented in three sections. The first section covers the p-median problem. The next section presents wireless sensor networks and mobile ad hoc networks. The final section covers potential machine learning approaches.

P-median

The p-median section of the literature review begins with papers that provide the foundation of the p-median problem. The subsequent papers reviewed demonstrate that machine learning techniques such as genetic algorithm and neural network are viable methods for obtaining p-median optimizations.

Hakimi (1964) is an early study on the problem of finding the center and median of a graph *G* where the edges and vertices are weighted. The problem is modeled as finding the ideal location of a switching center *S* in a communications network. The optimal location was where the length of connections from all vertices to *S* was minimized and this location was called the *absolute median* of the graph. Finding the *absolute center* of a graph is also discussed but is less relevant to this work and therefore not detailed here.

The work by Hakimi (1964) begins with definitions. The graph *G* is said to contain vertices *v* and branches *b* with weights h_i and w_i associated with vertices and branches respectively. The length of a path between two points on *G* is defined as the sum of the weights of the branches along the path and the distance between any two points $d(x, y)$ was defined as the minimum length path between the points. Given these definitions the absolute median was defined as a point y_0 "if for every point *y* on *G*" (Hakimi, 1964):

$$\sum_{i=1}^{n} h_i d(v_i, y_0) \leq \sum_{i=1}^{n} h_i d(v_i, y) \text{ (Hakimi, 1964)} \quad (7)$$

The work in Hakimi (1964) continues with providing a proof for the theorem that the absolute median of a graph must be at one of the vertices. The proof begins by substituting in the definition above and stating that for an arbitrary point x_0 on G where x_0 is not a vertex, there "exists a vertex v_m such that" (Hakimi, 1964):

$$\sum_{i=1}^{n} h_i d(v_i, x_0) \geq \sum_{i=1}^{n} h_i d(v_i, v_m) \text{ (Hakimi, 1964)} \quad (8)$$

The point x_0 is said to be on a branch $b(v_p, v_q)$of G. The distance of any vertex v_i

will be the minimum of the distance to either v_p or v_q plus the distance to x_0:

$$d(v_i, x_0) = \min[d(x_0, v_p) + d(v_p, v_i), d(x_0, v_q) + d(v_q, v_i)] \text{ (Hakimi, 1964)} \quad (9)$$

Equation (8) is substituted into equation (9) so that two equations result. It is then shown through additional substitution that both cases show that equation (7) is true. Therefore, it was proven that the absolute median was a vertex on the graph G (Hakimi, 1964).

The work in Hakimi (1964) is important in providing a foundation for future work on the p-median problem. It allowed for limiting the points considered as the median of a graph to the vertices. This work was extended to show that the solution to the p-median problem would consist of points on the vertices of a graph in Hakimi (1965) reviewed next.

Hakimi (1965) begins considering the same problem as above except in this case the problem was expanded from finding the ideal location for a single switching center S to multiple switching centers $S_1, S_2, \ldots, S_p$. The network was again considered a graph G with weights attached to the branches and vertices. Distance between points was defined the same as above. Given a set of points , $x_1, x_2, \ldots, x_p$, that represent the switching centers on G, X_p* is considered a p-median if for all possible X_p:

$\sum_{i=1}^{n} h_i d(v_i, X_p *) \leq \sum_{i=1}^{n} h_i d(v_i, X_p)$ (Hakimi, 1965)

The distance $d(v_i, X_p)$ was defined as the distance from v_i to the nearest member of X_p. This work also considered the problem of finding the p-centers that is less relevant to this work and not reviewed in detail.

Hakimi (1965) next walked through a proof building on the one in Hakimi (1964) that showed that within the set of vertices *V* on *G* "there exists a subset V_p* of *V* containing *p* vertices such that for every set of *p* points *X* on *G*" (Hakimi, 1965):

$\sum_{i=1}^{n} h_i d(v_i, X) \leq \sum_{i=1}^{n} h_i d(v_i, V_p *)$ (Hakimi, 1965)

Similar to the finding above, this proof showed that the set of points that make up the p-median are a subset of the set of vertices V of the graph. Therefore, when considering the solution for a location problem such as the ideal location for switching centers only graph vertices need to be considered (Hakimi, 1965).

Hakimi (1965) also presented a numerical method for finding the p-median. The sample problem was finding the optimal location of three switching centers within a network that contained *n* nodes. The solution included first creating an *n* x *n* distance matrix of the nodes on *G*. The distance matrix was then multiplied by the weight of the corresponding vertex. Next the sum:

$\sum_{r=1}^{n} \min(d_{ir}^*, d_{jr}^*, d_{kr}^*)$

was calculated for all possible values of *i*, *j*, and *k* where $(1 \leq i, j, k \leq n)$. The values where the above sum was a minimum were those that were selected for the switching centers (Hakimi, 1965).

Hakimi (1965) extended the work of Hakimi (1964) and proved that a p-median of a graph exists such that the points that make up the p-median are vertices of the graph. A numerical method was presented for finding the p-median of a small directed graph which showed the complexity of the problem. This work provided the foundation for subsequent heuristic-based approaches to finding the p-median of a graph.

Correa, Steiner, Freitas & Carnieri (2004) proposed using a genetic algorithm for solving a constrained version of the p-median problem. In their work the constraints involved solving a capacitated version of the p-median problem. P-median problems are generally presented as finding the optimal locations for a fixed number of facilities to minimize the distance to demand nodes on a graph. In typical p-median problems the facilities can supply an unlimited number of demand nodes, however in the capacitated version the supply from each facility is limited. An additional constraint that is similar to this work was that each demand node had to be associated with exactly one facility (Correa et al., 2004).

Correa et al. (2004) applied their work to a real-world scenario. Their problem consisted of 43 potential testing facilities that needed to supply the demand from 19,710 students. The goal was to choose the facilities that would minimize the distance from students' homes to their respective testing facility. It was predetermined that a subset of 26 facilities would be used to supply the demand. Correa et al. (2004) implemented a genetic algorithm to optimize the solution.

A typical genetic algorithm (GA) is a machine learning optimization algorithm that begins with random solutions, each one known as a chromosome. The GA then selects the best candidate chromosomes based on a fitness function and uses those to create the next generation

of chromosomes (Whitley, 1994). The chromosomes are combined using a probability-based crossover function that selects random members (genes) from two parent chromosomes and swaps them. Chromosomes are also altered through a mutation method that randomly changes genes within the chromosome. Mutation is also based on probability and the mutation probability is typically much lower than the crossover probability (Whitley, 1994). This process successively creates generations that are a better solution that the previous genera ration. This process generally continues for a predetermined number of generations or a specific fitness threshold is reached (Whitley, 1994).

The GA used in Correa et al. (2004) was modified to better fit the problem space. Each chromosome was a set of 26 indices that represented the ID of a particular facility. The fitness of each chromosome was measured by the total sum distance of all students from their assigned facility. The assignment process began by assigning each student to the nearest facility as long as that facility had capacity. Afterward the student was assigned to the next nearest facility with capacity until all students were assigned. After assignment was complete students not assigned to the nearest facility were exchanged and the total sum distance recalculated to determine if the exchange improved the solution. Improvements were retained and this process continued until the solution was no longer improved (Correa et al., 2004).

The crossover method implemented by Correa et al. (2004) removed duplicate facility indices so that no solution could have the same facility included more than once. A random number was used to determine the number of genes that were swapped. A heuristic was applied to the mutation method based on domain knowledge. A percentage of chromosomes were randomly selected and then each gene of the chromosome was replaced with a facility index not currently in the chromosome. If the change improved the fitness of the chromosome, then it was

retained. Finally, no chromosomes were added to the next generation if their fitness was no better than the worst performing chromosome of the previous generation (Correa et al., 2004).

Correa et al. (2004) tested their algorithm with and without the mutation heuristic and tested against the *tabu* search algorithm. In the tabu search algorithm a set of facilities were first randomly selected. Then the solution was iterated by either adding, dropping, or swapping facilities. Each operation was evaluated so that the facility that provided the minimum sum distance was used in the operation. This process was repeated for a predetermined number of iterations (Correa et al., 2004). The number of iterations for the two versions of the GA was adjusted to make the computational time roughly equivalent between both Gas and the tabu search. The version with heuristic mutation used 1,000 iterations. The heuristic mutation was very computationally expensive and the GA without it used 12,100 iterations with the runtime of the two GAs found to be similar. The tabu search used 150 iterations which resulted in a run time similar to the two GAs. The three algorithms had similar results based the total sum distance and the percentage of students assigned to the nearest facility. The GA with heuristic mutation performed the best with total distance of 45,999 km and 83% of students assigned to the nearest facility. The tabu search was next with 46,660 km total distance and 82% of students assigned to the nearest facility. The GA without heuristic mutation had a total distance of 47,313 km and 79% of students assigned to the nearest facility. The results showed that all three performed well and that the heuristic mutation is a viable method for improving the GA performance (Correa et al., 2004).

The work by Correa et al. (2004) showed that the GA can provide a good solution for the p-median problem. The constraint of having each student assigned to exactly one facility is

similar to this work and therefore may provide insight into a possible solution. There may be a heuristic mutation for the wireless sensor network problem space that improves the solution.

Domínguez & Muñoz (2008) proposed using a recurrent neural network to solve a constrained version of the p-median problem. Their work begins with reiterating that the p-median problem is NP hard and the authors list several heuristic methods from the literature. Next a brief description of neural networks (NN) is provided. It begins by stating that NNs are a viable solution for many different types of problem including pattern classification, clustering and combinatorial optimization. A binary artificial neuron was described as having an activation value that results in output of either 1 or 0. A recurrent neural network is described as a set of *N* artificial neurons connected by links that each have a weight associated with them. The concept of an energy function is introduced and provided as:

$E(t) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_i(t)x_j(t) + \sum_{i=1}^{N}\theta_i x_i(t)$ (Domínguez & Muñoz, 2008)

In the equation above, w_{ij} represents the weight of the connection between neurons *i* and *j*. The term "$x_i(t)$ is the activation value of neuron *i* at time *t*, and θ_i is the threshold value for the neuron *I*" (Domínguez & Muñoz, 2008). The energy function will either stay the same or decrease and when the energy function is stable it indicates the network has reached a minimum. It was also noted that these types of recurrent neural networks, also known as Hopfield networks, tend to settle into a local minimum versus a global minimum (Domínguez & Muñoz, 2008).

Domínguez & Muñoz (2008) next define the constraints applied to the p-median problem. Given a system with *n* demand nodes and *p* facilities the problem was to minimize:

$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{q=1}^{p} d_{ij}x_{iq}y_{jq}$ (Domínguez & Muñoz, 2008)

In the equation above, d_{ij} is the distance between the demand point i and the facility j. The terms x_{iq} and y_{jq} represented allocation variables and location variables respectively. The allocation and location variables were defined by:

$$x_{iq} = \begin{cases} 1 & \text{if } i \text{ is assigned to cluster } q, \\ 0 & \text{otherwise,} \end{cases}$$ (Domínguez & Muñoz, 2008)

$$y_{jq} = \begin{cases} 1 & \text{if } j \text{ is the center of cluster } q, \\ 0 & \text{otherwise,} \end{cases}$$ (Domínguez & Muñoz, 2008)

The first constraint was that each demand node was a member of only one cluster and was expressed as:

$$\sum_{q=1}^{p} x_{iq} = 1, \ i = 1, \dots, n,$$ (Domínguez & Muñoz, 2008)

The second constraint was that each cluster must contain exactly one facility and was expressed as:

$$\sum_{j=1}^{n} y_{jq} = 1, \ q = 1, \dots, p,$$ (Domínguez & Muñoz, 2008)

The constraints were included as a penalty value in the energy equation in order to express the problem as unconstrained. This yielded the energy equation below:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{q=1}^{p} d_{ij} x_{iq} y_{jq} + \lambda_1 \sum_{i=1}^{n} \left(1 - \sum_{q=1}^{p} x_{iq}\right)^2 + \lambda_2 \sum_{q=1}^{p} \left(1 - \sum_{j=1}^{n} y_{jq}\right)^2$$

(Domínguez & Muñoz, 2008)

The $\lambda_i > 0$ parameters represent the weight of the penalty term and it is noted that finding proper values for the penalty weights is an "important problem associated with this approach" (Domínguez & Muñoz, 2008).

In order to avoid the problem of tuning the penalty parameters, Domínguez & Muñoz (2008) structured their neural network in such a manner so that the architecture of the network enforced the constraints. This was accomplished through constructing the network as a set of disjoint groups where for *n* demand points there were *n* groups with only one neuron in each group allowed to be active at any time. Similarly, for *p* facility locations there were *p* groups also with only one neuron allowed to be active. In this manner, the constraints were satisfied through the network architecture and the penalty terms could be removed from the energy equation which resulted in the simplified equation:

$E = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{q=1}^{p} d_{ij} x_{iq} y_{jq}$ (Domínguez & Muñoz, 2008)

The structure of the disjoint groups is shown below in Figure 4:

$$
\begin{array}{ccccc}
x_{11} & x_{12} & \cdots & x_{1p} & \rightarrow \text{ group 1} \\
x_{21} & x_{22} & \cdots & x_{2p} & \rightarrow \text{ group 2} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
x_{n1} & x_{n2} & \cdots & x_{np} & \rightarrow \text{ group n} \\
y_{11} & y_{12} & \cdots & y_{1p} & \\
y_{21} & y_{22} & \cdots & y_{2p} & \\
\vdots & \vdots & \ddots & \vdots & \\
y_{n1} & y_{n2} & \cdots & y_{np} & \\
\downarrow & \downarrow & & \downarrow & \\
\text{group n}+1 & \text{group n}+2 & & \text{group n}+\text{p} &
\end{array}
$$

Figure 4. Disjoint Groups (Domínguez & Muñoz, 2008)

Domínguez & Muñoz (2008) implemented two different algorithms for updating the network. Group-parallel dynamics (NA-G) updated one group at a time while layer-parallel dynamics (NA-L) updated all groups in a layer at the same time. The two algorithms were tested on a set of p-median problems with known optimal solutions. It was found that the results from both were relatively poor and this was discovered to be due to the initial selection of the facility locations. An additional scattering algorithm was added to NA-L to set the initial locations of the facilities and this improved the performance of the algorithm but also significantly increased computational overhead. The new algorithm, NA-L^+, was tested against the known heuristic method variable neighborhood search (VNS). The computational time of VNS was limited to be similar to the time of NA-L+. With this constraint, NA-L+ obtained superior optimization on almost all of the test scenarios. This showed that the neural network approach was a viable

method for finding the p-median and that it was able to do so in less time computationally (Domínguez & Muñoz, 2008).

This work presented in (Domínguez & Muñoz, 2008) is important for this work since the constraints are identical and the authors were able to implement a machine learning method to find optimal p-median solutions. IT is unknown if the VNS algorithm would ultimately have found better solutions given more run time and will be interesting to compare the solution proposed in this work to VNS in terms of solution quality and run time.

Wireless sensor networks and MANETs

As noted above wireless sensor networks (WSNs) represent a special case of mobile ad hoc networks (MANETs). There are two primary differences between WSNs and MANETs. The first is that in MANETs the nodes are typically mobile whereas in WSNs they are not. There may be some node mobility in WSNs such as sensors floating in the ocean on currents, but generally this movement is not significant as compared to the mobility of MANET nodes. The other difference is the existence of the base-station or sink in a WSN. MANET nodes typically only communicate with each other but WSNs send information to a sink that then transmits the information to other systems where it is consumed. Otherwise the two networks present many of the same problems including limited battery power of devices and the requirement for the network to be set up quickly without pre-existing infrastructure. Therefore, most of the solutions presented for MANETs are also applicable to WSNs.

Drugan et al. (2011) proposed implementing a clustering method based on work done in identifying communities such as social networks. The reasoning behind this approach was that clusters could be identified based on routing information and therefore no additional

communication overhead would be added for cluster discovery and maintenance. It is acknowledged by the authors that sparse MANETs pose additional problems versus systems with large numbers of data points and therefore the methods used for larger numbers of objects must be modified for use with sparse MANETs (Drugan et al., 2011).

The approach is based on the underlying assumption that areas of nodes with dense connections to each other represent communities or clusters and that individual clusters will have fewer connections between them (Drugan et al., 2011). This is relevant to the work proposed in this dissertation since this approach, although relying on network connections instead of node location is a method to measure of node density. Drugan et al. (2011) also use k-medoids as an algorithm for comparison and point out the shortcoming of k-medoids that it requires a predetermined number of clusters in order to operate. The authors implement several different community detection algorithms make minor adjustments to the algorithms to account for detection of clusters in sparse data. The algorithms implemented included a modularity- based algorithm: Newman and Griven Community Detection, random walk algorithm: vanDongen Community Detection, and potts-based: Reichard and Bornholdt Community Detection (Drugan et al., 2011).

Drugan et al. (2011) explain the different types of routing protocols typically used in MANETs. There are reactive protocols that perform route discovery when communication is required and proactive protocols that maintain routing information and update it periodically. Optimized Link State Routing (OLSR) is a proactive protocol. OLSR is ideal for “large and dense mobile networks” (Clausen & Jacquet, 2003). Individual nodes maintain routing tables of network topology that is shared by other nodes. Using this information each node can use the information it has stored locally to route packets. OLSR also makes use of nodes that are

designated as multipoint relays (MPRs). A node designates its one-hop neighbors that it can establish two-way communication with as MPRs. MPRs broadcast to the network that they can reach the nodes that have designated them as MPRs and also are responsible for network control traffic. In this manner OLSR is able to reduce the amount of flooding traffic required to maintain network routing paths (Clausen & Jacquet, 2003).

The algorithms were all evaluated using Global Mobile Information System Simulation Library network simulation. The OLSR protocol was used along with both static and mobile network nodes. It was also assumed that 25% of the nodes would send communications every 5 seconds. Forty nodes were used in the simulation in an area of 800m x 600m and the mobility of the dynamic nodes varied from 1m/s to 7m/s and a node transmission range of 100 m (Drugan et al., 2011).

The Silhouette index was used to measure the quality of clustering. It was developed by Rousseeuw (1987) as part of a method to graphically determine the quality of clusters and also help determine if the appropriate number of clusters was created from a set of points. The creation of a silhouette depended on the calculation of an index value denoted $s(i)$. Other values calculated were the dissimilarity of a node to its associated cluster, $a(i)$, and dissimilarity with the next nearest cluster, $b(i)$. In Figure 5 below $a(i)$ is the average distance of the point i to the other points in the cluster of which it is a member, in this case cluster A. The value $b(i)$ is the average distance to points in the next nearest cluster, in this case cluster B (Rousseeuw, 1987).

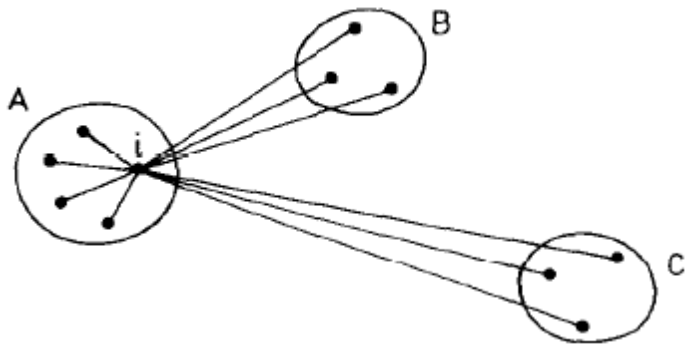

Figure 5. Clusters (Rousseeuw, 1987)

The value for *s(i)* can then be calculated using Equation (10) below. Per Equation (10), *s(i)* will vary from values of -1 to 1, with 1 indicating that the node *i* is well clustered while a value of *s(i)* closer to -1 indicates poor clustering (Rousseeuw, 1987).

$$s(i) = \frac{b(i)-a(i)}{max\{a(i),b(i)\}} \text{ (Rousseeuw, 1987)} \qquad (10)$$

Drugan et al. (2011) used the equation above to calculate *s(i)* for each node and then found the average value of *s(i)* for each cluster and the average value of *s(i)* for the entire network. It was found that generally the community detection methods used were able to obtain quality clustering. Negative values which indicated poor clustering were explained as due to disconnected nodes (Drugan et al., 2011).

The approach used in Drugan et al. (2011) is interesting, especially since it does not introduce communication overhead into the network. However there is no discussion of increased resource usage for calculation of the clusters. Also each of the algorithms used had to be adjusted with constraints on cut off or adjustment of constants. It would be interesting to see if these adjustments were valid across many different scenarios as far as the number of nodes and mobility or needed to be readjusted for each scenario.

Sett & Thakurta (2015) proposed an approach to determine the ideal number and location of cluster heads of a network using a genetic algorithm (GA). The indicators used to determine fitness for the GA were minimization of number of clusters, maximization of coverage area, even distribution of cluster heads, and uniformity in the number of nodes in each cluster. The use of the multiple criteria for fitness made this a multi-objective optimization problem and therefore Pareto dominance sorting was used to rank the fitness of the members.

In a multiple-objective optimization problem it is unlikely there is one best solution. One method of resolving this problem is to create a set of optimal solutions. A Pareto front is a set of optimal solutions that are termed non-dominated. A solution is said to dominate another solution if it is better in at least one objective and no worse in all other objectives (Mishra & Harit, 2010). The set of non-dominated solutions was used as the parents for the next generation in the GA (Sett & Thakurta, 2015).

The system was configured with a value of between 40 and 100 nodes with the lower limit estimated for effective functioning of the GA and the upper limit to cap executional complexity. Crossover and mutation were used with the successful parents to create each subsequent generation (Sett & Thakurta, 2015).

Sett & Thakurta (2015) evaluated their method through experimentation and found that it was not possible to plot an ideal solution using the GA alone due to the conflicting nature of the fitness indicators. Pareto dominance was used with individual solutions for the indicators to combine results and attempt to find over best fit solutions (Sett & Thakurta, 2015). This approach is interesting but likely not ideal for an actual MANET. As seen in other studies from the literature, MANETs require frequent rebalancing due nodes joining and leaving along with mobility. Executing the GA algorithm for each rebalancing would be too expensive and

therefore not feasible in practice. However if a GA could be used to locate an ideal solution for a particular configuration or at a particular instant in time this solution could be used as a benchmark for other potential solutions. This approach was explored as part of this topic dissertation work .

A genetic algorithm (GA) was proposed for clustering the nodes of a MANET and it was shown that the problem of finding the optimal set of cluster heads was NP-hard (Cheng, 2012). Therefore the author proposed a system using genetic algorithms (GA) with a high mutation rate to find the best approximate solution (Cheng, 2012).

The GA in Cheng (2012) was set up so that each node in the network was assigned an ID number, and those numbers were used to generate the chromosomes for the GA. Once an ID was selected as part of the chromosome, any nodes within one hop of that node would not be allowed as part of the chromosome. Nodes selected for the chromosome were to be evaluated as cluster heads. The fitness function evaluated each chromosome based on how evenly balanced the cluster heads were in terms of how many member nodes were in each cluster. The selection scheme used was pair-wise tournament with a size of 2. The xOrder1 method was used for crossover while gene swapping was used for mutation (Cheng, 2012).

Cheng (2012) explained while standard mutation levels function well in static environments they do not work in more dynamic environments such as MANETs. In order to deal with the changing topology of MANETs, Cheng (2012) employed a hyper mutation genetic algorithm (HMGA). The idea was that as the MANET topology was more dynamic then the mutation level would be increased and as the MANET was more stable the mutation would decrease. Two models of HMGA were developed and tested. One was termed high low (hlHMGA) because it worked by setting the mutation level high for first half of the set of

generations and low for the second half. The other model was termed gradual (grHMGA) since it started at a high level of mutation after a change and then gradually reduced the mutation until the next change (Cheng (2012).

The GA models were tested on a simulated MANET in a 200 unit square area with 100 nodes and 50 unit transmission range. Rather than using mobility to simulate topology changes, network nodes were randomly turned off or on within a predetermined interval. It was found that both models performed better when the range of high to low mutation was small; 0.1 as compared to 0.4 and 0.7. It was also found that above 80 generations the hlHMGA performed better than the grHMGA (Cheng, 2012).

The clustering approach used in Cheng (2012) was focused on load balancing the changing topology of MANETs. A drawback of this approach for MANETs was the need for a central authority to execute the HMGA. However this issue can be mitigated in a wireless sensor network by offloading the processing to a connected system. Also, running the HMGA likely uses more network resources as compared to other solutions since it is iterative and appears to converge at approximately 200 generations. It would have been interesting to see how the HMGA solution compared to other solutions from the literature in terms of clustering overhead.

Peiravi, Mashhadi & Hamed Javadi (2013) implemented a multi-objective genetic algorithm (GA) in order to cluster WSNs. The algorithm was named multi-objective two-nested genetic algorithm (M2NGA). M2NGA used two nested genetic algorithms; the first was intended to optimize the clusters of the network and the second level optimized the transmissions within each cluster (Peiravi, Mashhadi & Hamed Javadi, 2013).

Peiravi, Mashhadi & Hamed Javadi (2013) encoded the chromosomes as an $m \times 2$ matrix where m was the number of nodes. This resulted in two values for each node where the first

value was the cluster the node was a member of and the second indicated if the node was a cluster head. Instead of random, the initial population of chromosomes was created such that all the nodes of a cluster were within transmission range of each other and a cluster head was randomly chosen from within each cluster (Peiravi, Mashhadi & Hamed Javadi, 2013).

Peiravi, Mashhadi & Hamed Javadi (2013) defined the lifetime of their network as the time until the first node failed due to power loss. The fitness function that calculated the fitness of the routing within each cluster was:

$$fitness_i = \exp(-0.1 \times (\max(Delay_i) - 1)) \times \frac{\max(Energy_{pop})}{\max(Energy_i)}$$

(Peiravi, Mashhadi & Hamed Javadi, 2013)

In the equation above $Delay_i$ represented the number of hops, $Energy_{pop}$ was the energy consumption of the chromosomes of the populations, and $Energy_i$ was the energy consumption of chromosome *i*. The exponential function was used to reduce the impact of chromosomes with large delays. The functions used to calculate energy usage are provided below:

$E_{T_{ij}} = k \times E_{elec} + k \times E_{amp} \times d^2 \quad E_R = k \times E_{elec}$ (Peiravi, Mashhadi & Hamed Javadi, 2013).

The first equation represented the energy required to transmit *k* bits from *i* to *j* and the second represented the energy to receive *k* bits. The terms E_{elec} and E_{amp} represented the energy used in the circuit and amplifier respectively and d^2 was the distance between *i* and *j* (Peiravi, Mashhadi & Hamed Javadi, 2013).

Peiravi, Mashhadi & Hamed Javadi (2013) compared M2NGA to other GA based algorithms from the literature. Their simulation consisted of 40 nodes in a 100m by 100m area and was implemented in a MATLAB environment. It was found that the nested GA clustering solution consumed less energy sending bits from nodes to the sink. The authors acknowledged

that the nested GA solution used more resources computationally but noted that this was not an issue because the calculations were carried out at the sink which is typically connected to an unlimited power supply (Peiravi, Mashhadi & Hamed Javadi, 2013).

The work done by Heinzelman, Chandrakasan & Balakrishnan (2002) is a seminal study on clustering WSNs and has been cited over 10,000 times. The authors developed a cluster-based routing system called the "low-energy adaptive clustering hierarchy (LEACH)". The study begins by listing the qualities of a successful WSN. These included "ease of deployment", "system lifetime", latency", and "quality" (Heinzelman, Chandrakasan & Balakrishnan, 2002). Quality is included to differentiate WSNs from typical networks in that the important aspect is an overall picture of the conditions being monitored. Therefore if data from a node is not received successfully it is mitigated by the fact that there will typically be other nearby nodes relaying similar data (Heinzelman, Chandrakasan & Balakrishnan, 2002).

One of the key ideas in LEACH was the aggregation of data at the cluster heads in order to reduce network traffic. An assumption made in the development of LEACH was that all nodes were within transmission range of the sink. The authors divided the operation of the WSN into rounds. Each consisted of clustering the nodes, transmission of messages to cluster heads and transmission from cluster heads to the sink (Heinzelman, Chandrakasan & Balakrishnan, 2002).

Heinzelman, Chandrakasan & Balakrishnan (2002) used a predetermined number of clusters for each round. The clustering algorithm operated in a distributed manner, with the nodes calculating the cluster heads for each round. The goal of the clustering algorithm was to evenly distribute the number of nodes managed by each cluster head. Another feature of the

clustering algorithm was to elect nodes as cluster heads that had not recently been cluster heads so that the extra energy required for cluster heads was evenly distributed among all of the nodes. Toward this goal the remaining energy of a node was used as part of the calculation to determine which nodes became cluster heads. The authors also showed that in order for the cluster selection to function each node needed to be aware of the optimal number of clusters. This was shown that it could be accomplished by each node sending “hello” messages a predetermined number of hops and then using the number of messages received by a node to approximate the total nodes in the network. Later work showed that the optimal number of clusters could be determined using only the number of nodes in the WSN and the area covered by the WSN (Heinzelman, Chandrakasan & Balakrishnan, 2002).

Heinzelman, Chandrakasan & Balakrishnan (2002) used signal strength in order to determine which cluster a node was a member of. Once the cluster heads were identified, each would send an announcement that it was a cluster head. It was assumed the nodes could measure the signal strength of the received message and would join the cluster associated with the announcement of highest signal strength (Heinzelman, Chandrakasan & Balakrishnan, 2002).

Heinzelman, Chandrakasan & Balakrishnan (2002) also described a modified version of LEACH called LEACH-C. This version used a centralized algorithm to select the cluster heads. The average energy of all nodes was calculated at the sink and nodes below this threshold could not be selected as cluster heads. A simulated annealing algorithm was implemented at the sink to calculate the optimal cluster configuration (Heinzelman, Chandrakasan & Balakrishnan, 2002).

Heinzelman, Chandrakasan & Balakrishnan (2002) tested LEACH against LEACH-C and MTE using the network simulator *ns*. Experiments were run using 100 nodes in a 100m square area. It was found that both LEACH and LEACH-C were more energy efficient than MTE and that LEACH-C was about 40% more efficient than LEACH in terms of "data per unit energy" (Heinzelman, Chandrakasan & Balakrishnan, 2002).

Heinzelman, Chandrakasan & Balakrishnan (2002) acknowledged that the assumption that all nodes are within transmission range of the sink and each may not be valid and provided a discussion on possible changes to LEACH to account for that. The authors suggested a hierarchical approach to clustering or multi-hop architecture. However implementing a multi-hop or hierarchical clustering configuration would either limit the possible choices for cluster heads or require more than optimal clusters in order for there to be complete communication from all nodes to the sink. It would be interesting to see under what conditions it is preferential to have more clusters than optimal versus fewer options for cluster heads.

Kim, Seok, Choi, Choi, & Kwon (2005) Performed a study involving a WSN architecture with multiple sinks. The goal of the research was to use traffic engineering to approach two different types of multi-sink problems. The first was where the sinks were predetermined and in fixed locations and the second was one in which the number of sinks was fixed but the locations could be optimized. The primary focus of the work was the second problem. The authors also provide a general description of WSNs and note that the number of sinks is typically limited due to the significantly higher cost of sinks as compared to typical WSN nodes (Kim et al., 2005).

Kim et al. (2005) implemented a linear programming formulation focusing on fairness and lifetime. Fairness was defined as assuring that each node could communicate a minimum

volume of data to a sink. Lifetime was defined as the duration of the network until all of the nodes ran out of energy. Assumptions made by authors included that the idle time energy usage of nodes was negligible, sinks had effectively an unlimited power supply, and sinks did not transmit data. The authors acknowledged that the problem involving the optimal placement of multiple sinks is NP-hard. Therefore the work was limited to a system of less than 30 nodes. A network of 20 nodes was used for testing and comparison to another algorithm (Kim et al., 2005).

Kim et al. (2005) compared their solution to an algorithm called multi-sink aware Minimum Depth Tree (m-MDT). The comparison was done in a simulated area of 200m square using 20 nodes with varying number of sinks. The number of sinks was fixed for each test, starting at one and incrementing by one up to five. It was shown in experiments that both fairness and lifetime were improved with the linear formulation as compared to m-MDT (Kim et al., 2005),

The paper by Kim et al. (2005) demonstrates that WSNs with multiple sinks is an interesting problem worthy of further study. Multiple network paths were used to decrease the overall energy use and improve the fairness of the WSN. The placement of the multiple sinks was also determined based on reducing network energy consumption. It would be interesting to implement a node clustering scheme along with the node placement problem.

Deep Learning

The final section of the literature review presents papers on deep learning. It is intended that along with the more traditional genetic algorithm approach seen in the previous sections of

the literature review that a deep learning approach maybe attempted to solve the constrained p-median problem presented in this work is time permits.

The work by LeCun, Bengio & Hinton (2015) provided an excellent overview of deep learning. It begins by stating the limitations of conventional machine learning algorithms. Particularly that they often required domain expertise and significant work to extract and transform the features to be usable for machine learning. Deep learning methods are contrasted as a representation learning method. Data is represented at multiple levels with each level transforming the data so that complex problems can be modeled. The image recognition problem is given as an example. The initial layer may only identify edges and then the subsequent layer may identify collections of edges in relative locations. The next layer may assemble the collections into larger combinations and the layer after that may recognize the collections or assemble them into larger collections. It is noted that the important idea is that the layers are not designed as may have been required in earlier methods, but learn in response to a "general-purpose learning procedure" LeCun, Bengio & Hinton (2015).

LeCun, Bengio & Hinton (2015) next discussed supervised learning. The authors noted that typically in supervised learning stochastic gradient descent is used to train the algorithm. This was described as using training examples as inputs to the network and then calculating the output and errors. Next the gradient of the errors is used to adjust the weights of the network. It was noted this type of classifier performs well on linear classification problems but nonlinear problems such as image recognition require a different approach. Deep learning provides an architecture to solve such problems through a stack of layers that perform nonlinear mappings from input to output (LeCun, Bengio & Hinton, 2015).

LeCun, Bengio & Hinton (2015) discussed the backpropagation algorithm and extend it to deep learning. The authors note the known tendency of the backpropagation algorithm to settle into local minima. Recent work has shown that this is not necessarily an issue since there are typically many such local minima and all have approximately the same value of the objective function (LeCun, Bengio & Hinton, 2015).

LeCun, Bengio & Hinton (2015) discussed convolutional neural networks (ConvNets) next, noting that they are especially good at processing data that is composed of multiple arrays such as color images. ConvNets architecture is typically composed of alternating layers of convolutional layers and pooling layers. The convolutional layers detect individual features and the pooling layers combine those features into more complex features. ConvNets are currently in use for many tasks involving images included facial recognition and self-driving cars. A typical ConvNet architecture is shown below in Figure 6, the data flow is from input at the bottom of the figure up.

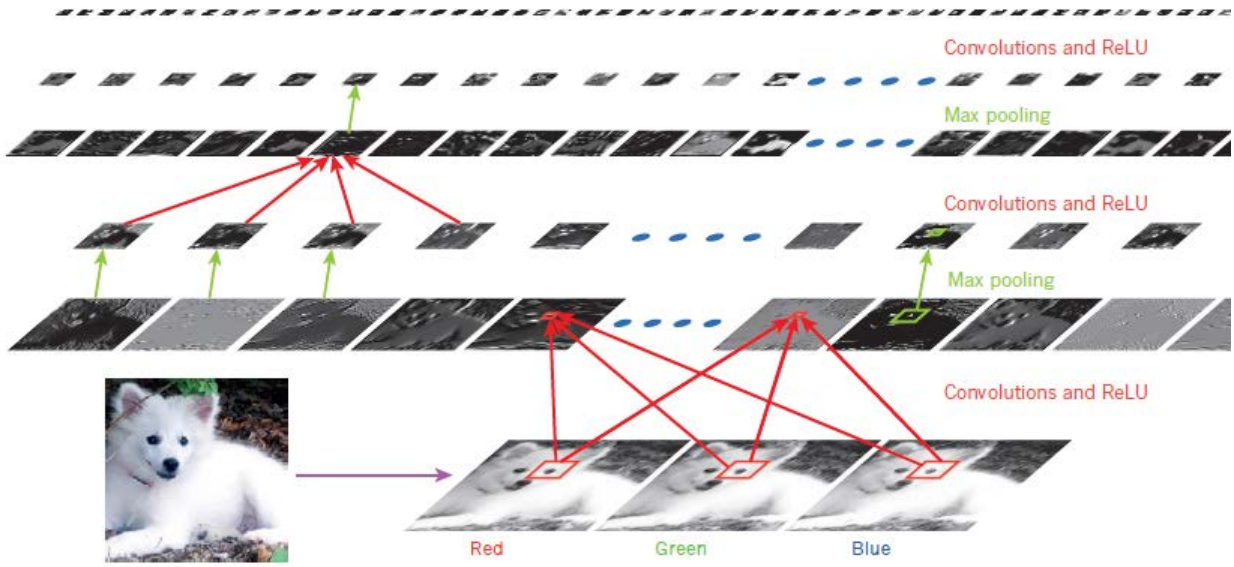

Figure 6. ConvNet (LeCun, Bengio & Hinton, 2015)

Recurrent neural networks (RNNs), which are applicable to problems where the input is sequential were discussed in the work by LeCun, Bengio & Hinton (2015). The hidden layers in a RNN retain information on the history of the network. The backpropagation algorithm is typically used to train RNNS, but there is difficulty in that the gradient grows or shrinks with each iteration and therefore tend to "explode or shrink" (LeCun, Bengio & Hinton, 2015). Despite the difficulty in training RNNs, with recent advances they are useful in predictions such as the next word in a sentence. However, it has been shown that RNNs have difficulty in retaining information and a solution has been proposed as long short-term memory (LSTM). LSTMs use special hidden neurons called memory cells that accumulate information. These networks have shown promise not only in remembering information but also in tasks that require reasoning (LeCun, Bengio & Hinton, 2015).

LeCun, Bengio & Hinton (2015) presented an excellent overview of the progression form early neural networks to deep learning and the potential future direction of deep learning. Although the focus of the paper is on supervised learning, the authors expect unsupervised learning to become more important in the near future (LeCun, Bengio & Hinton, 2015).

Krizhevsky, Sutskever & Hinton (2012) presented a method for classifying images using convolutional neural networks (ConvNets). The paper begins explaining the difficulty in image recognition that arises from having relatively limited training data. It was stated that the best type of network to overcome this limitation is one that works with prior knowledge. ConvNets were chosen to this capacity as well as having fewer parameters and being easier to train than standard feedforward networks Krizhevsky, Sutskever & Hinton (2012).

The architecture of the network used in Krizhevsky, Sutskever & Hinton (2012) is shown below in Figure 7.

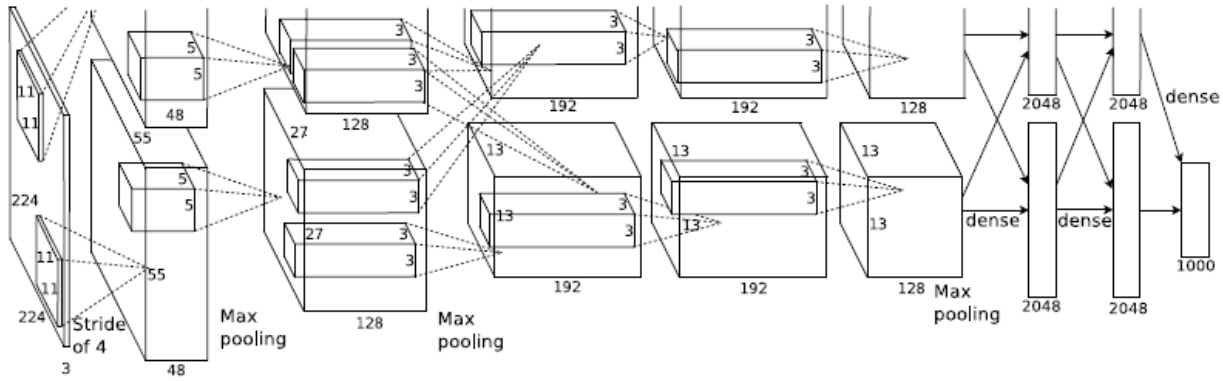

Figure 7. ConvNet (Krizhevsky, Sutskever & Hinton, 2012)

As shown the ConvNet was run in two parallel paths only communicating with each other at certain layers. This was to accommodate the hardware. Two GPUs were used with each one processing one of the paths. The first five layers of the network were convolutional with the last three being fully connected. Rather than use the sigmoid function or *tanh* for neuron output as is common, Krizhevsky, Sutskever & Hinton (2012) used a non-linear function:

$f(x) = \max(0, x)$ (Krizhevsky, Sutskever & Hinton, 2012)

Neurons that use non-linear functions were referred to as Rectified Linear Units (ReLUs) and found to converge six times faster than *tanh* neurons. The pooling layers in the ConvNet shown above used overlapping pooling as a method to reduce errors. Pooling was explained as mapping a neighborhood of pixels size $z \times z$ into a grid of units s pixels apart. In typical pooling $s = z$ however in this work $s = 2$ and $z = 3$ produced lower error rates (Krizhevsky, Sutskever & Hinton, 2012).

Krizhevsky, Sutskever & Hinton (2012) implemented additional techniques to improve the ConvNet performance. Data augmentation, increasing the number of training samples was done creating mirror images of the input images. Also, a technique known as *dropout* was employed to reduce overfitting of the model. Typically, dropout is implemented by setting the output value of neurons to 0 with a probability of 0.5. However, Krizhevsky, Sutskever & Hinton (2012) approximated this through multiplying by 0.5 the output of all neurons in the first two fully connected layers of the network.

The ConvNet presented in Krizhevsky, Sutskever & Hinton (2012) was tested against a subset of the ImagNet image database at a 2010 competition. The ConvNet achieved an error rate of 37.5% which was better than the stat-of the-art at the time. It was also noted that the depth of the network was important since the performance of the network was degraded if any of the layers were removed. It was also postulated that the performance of the ConvNet could be improved by pre-training some of the layers or by increasing the size of the network (Krizhevsky, Sutskever & Hinton, 2012).

Summary

As shown in the literature review above, there were many examples of attempted solutions of the p-median problem. It is evident that this is a problem worthy of continued study. Additionally there are examples of work done in clustering the nodes of wireless sensor networks (WSN) in order to prolong the lifetime of the network. Again this shows the value of additional work in this area. As noted in the introduction, the p-median problem can be constrained so that it closely resembles the problem of clustering WSN nodes. Therefore this research is applicable to both areas of study.

Methodology

Overview

As shown in the literature, genetic algorithms (GAs) have shown promise in creating good solutions for the p-median problem. Therefore, GAs were the primary focus of this work. In future work additional deep learning methods may be explored. Although other solutions to the p-median problem using GAs exist (Correa et al., 2004), the approaches had to be modified to accommodate the additional challenges associated the wireless sensor networks (WSNs). For example, in Correa et al. (2004) the authors knew at the start of the problems how many facilities were being used to satisfy the demand of the remaining points. In the WSNs the optimal number of clusters is unknown at the start of the problem.

Determining the ideal number of clusters in a WSN required the chromosomes of the GA to represent all of the nodes available in the network. Therefore, recalling from the problem statement that the problem begins with a set of nodes $\mathrm{V} = \{v_i \mid 1 \leq i \leq n\}$, one gene *p* was associated with exactly one node *n* using an ID for the node corresponding to the index of the gene *p*. This is similar to the approach in (Peiravi, Mashhadi & Hamed Javadi, 2013) except their approach involved a matrix of 2 values for each gene, one represented the cluster a node was a member of and the other indicated if the node was a cluster head (*CH*). The intention in this work was to only use the gene to indicate if a node is a cluster head, and then use proximity to the cluster heads in order to determine cluster members. There was additional work to determine the best method to deal with nodes that are within transmission range of more than one *CH*.

Chromosome Encoding

In this work the chromosomes of the GA were encoded such that each gene corresponded to one node of the network. Therefore each chromosome was represented by a $1 \times n$ binary matrix:

$$MA_{chromosome} = [g_1, g_2, g_3, \dots g_n].$$

The encoding of the genes indicated which nodes were cluster heads and which were member nodes according to equation (11) below.

$$g_i = \begin{cases} 1\ ;\ v_i \in CH \\ 0\ ;\ v_i \in B \end{cases} \tag{11}$$

As shown a gene had a value of one if the corresponding node was a cluster head and zero if it was a member node.

A sample chromosome from a network with 6 nodes is shown in Figure 8 below.

	0	0	1	0	1	1
gene:	1	2	3	4	5	6

Figure 8. Typical chromosome

In Figure 8 each box represents a gene within the chromosome. Each gene represents a node on the network. A “0” indicates that this node of the network is a member node while a “1” in the box indicates that the corresponding node is a cluster head. A sample network that the above chromosome could represent is shown below in Figure 9.

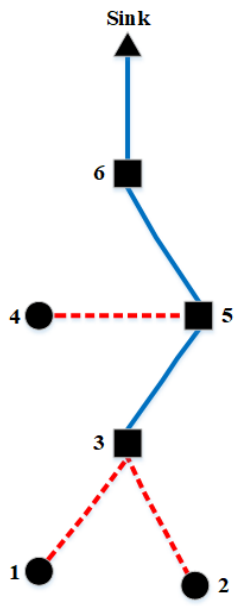

Figure 9. 6-Node WSN

In Figure 9 above the nodes shown as squares are cluster heads and are also indicated as such by a “1” in their corresponding gene shown in Figure 9. The circle nodes are member nodes and similarly are indicated by a “0” in their corresponding gene in Figure 9. The triangle node in Figure 9 is the sink where ultimately the data from all of the nodes is transmitted.

The configuration shown in Figure 9 represents a feasible network, one where all of the nodes can communicate with the sink. The red (dashed) lines indicate communication from the member nodes to their respective cluster heads. The blue (solid) lines indicate the overlay network which is the communication path from the cluster heads to the sink. The paradigm of the two graphs, a *RED* (dashed) one indicating connectivity from member nodes to cluster heads, and a *BLUE* (solid) one indicating the overlay network connecting cluster heads to the sink, was used to simplify the problem description and help formulate solutions and present solution descriptions. The chromosome repair method that was developed and tested as part of this work

functioned by creating sufficient square nodes so that every circle node was within range of a square node and then identified the optimal blue (solid) path to the sink.

In order to evaluate each chromosome a fitness function was used. In this work the target function shown in equation (6) was used to evaluate the fitness of each chromosome. As noted above, cluster heads were indicated via the genes of the chromosomes and cluster membership determined via proximity to cluster heads. The resulting topology was evaluated with the fitness function. Network configurations that did not satisfy the constraints detailed in the problem statement received low fitness scores. However, it was found be possible to repair chromosomes in order to satisfy the constraints and thereby improve their fitness score. Heuristics were developed to facilitate this repair process. It was also found to be valuable to simplify the target function for the fitness calculation in order to reduce computational time. Based on the fitness evaluation there are several potential methods to choose which chromosomes to retain to create the next generation. For example the Elite method chooses the chromosomes that have the best fitness score to be used as parents to create the next generation. Tournament chooses random chromosomes and the one out of the chosen group with the best fitness score becomes a parent for the next generation. The selection methods tested are detailed in the specific methods below. The methods evaluated in this work included Roulette-Wheel, Rank, and Tournament (Bayrakli & Erdogan, 2012) as well as others that appeared promising.

Crossover and Mutation

Additional methods used with GAs are crossover and mutation (Whitley, 1994), both of which were implemented in this work. Crossover was implemented using a random number for

the crossover position. The two parent chromosomes were selected using one of the methods noted above.

The formulas used for parent selection are described in detail in the specific methods section. Once the parents were selected, crossover was implemented to create the next generation of chromosomes. A crossover point $cp|(1 \leq cp \leq n)$ was selected randomly. Given two parent chromosomes *p* and *q*, crossover created two new chromosomes *x* and *y* using the equations below.

$$x_i = \begin{cases} p_i & i \leq cp \\ q_i & i > cp \end{cases}$$

$$y_i = \begin{cases} q_i & i \leq cp \\ p_i & i > cp \end{cases}$$

In the equations above, x_i represents the gene at position *i* in chromosome *x*.

Given two chromosomes, C1 and C2 below, assume the randomly selected crossover point is the third gene.

C1:	0	0	0	0	0	0
C2:	1	1	1	1	1	1

The two new chromosomes, NC1 and NC2, that result after crossover are shown below.

NC1:	0	0	0	1	1	1
NC2:	1	1	1	0	0	0

Mutation was performed by randomly selecting a gene and then changing its value. Given chromosome C3 below, assume the second gene is the one randomly selected for mutation.

C3:

0	0	0	0	0	0

The new chromosome that results is shown as NC3 below.

NC3:

0	1	0	0	0	0

Mutation used a probability to determine if it occurs and this probability was determined experimentally. Mutation was performed by selecting a gene and then changing its value. After new chromosomes were created using crossover, mutation was applied to the chromosome. First the mutation point $mp|1 \leq mp \leq n$ was selected randomly. Then the chromosome was mutated using the equation below.

$$x_i = \begin{cases} x_i & i \neq mp \\ x_i \oplus 1 & i = mp \end{cases}$$

In the equation above x represents a chromosome and x_i represents the gene at position i in x. Given chromosome C3 below, assume the second gene is the one randomly selected for mutation.

C3:

0	0	0	0	0	0

The new chromosome that results is shown as NC3 below.

NC3:

0	1	0	0	0	0

Helper Functions

As noted previously a network partition was considered feasible if and only if the *RED* and *BLUE* graphs were feasible. The *RED* graph was only feasible if all of the nodes in the *RED* graph were connected to a *CH*. This determination was simplified by utilizing a subset of $MA_{Connect}$. The subset MA_{red} included all columns that were not *CHs* and only rows of $MA_{Connect}$ where $v_i \in CH$ resulting in an $m \times (n - m)$ matrix. With this arrangement, summing a column of MA_{red} indicated that the corresponding node was within range of a *CH* if the sum was greater than or equal to one. Using MA_{red}, the Boolean function RED() was simplified as:

$$RED(B) = \begin{cases} true \ ; \ \ if \ \forall \ b \in B: \sum_{i=1}^{m} MA_{red}(i, b) \geq 1 \\ false \ ; \ \ otherwise \end{cases} \tag{12}$$

A binary matrix was created to simplify the calculation of *BLUE* graph feasibility. MA_{blue} was an $m \times (m + c)$ matrix with each column representing a *CH* or a sink node. The values in MA_{blue} were provided by equation (13) below.

$$MA_{blue}(v_i, v_j) = \begin{cases} 1 \ ; \ d(v_i, v_j) \leq R \\ 0 \ ; \ d(v_i, v_j) > R \end{cases} \tag{13}$$

Specific Methods

The hypothesis tested in this work was determining if a genetic algorithm (GA) could be used to successfully solve the p-median problem that has been constrained to emulate the environment of a wireless sensor network (WSN). The work to prove this hypothesis proceeded

in phases with each phase building upon the previous one. The intent was that the work would begin with a simplified problem statement. Once an algorithm had been shown to be successful for the simplified version, complexity was added back in. Each phase increased the complexity and the algorithm was modified as required to be successful with the more complex problem. Working in this manner allowed for incremental progress toward the goal of the original problem statement. There were instances where the algorithm from the previous phase could not be successfully modified to function with the increased complexity. In these cases the previous version of the algorithm was reconsidered with the additional information from the new phase. While this approach did result in some rework, generally this cumulative approach allowed the work to build upon itself until it was shown that the hypothesis was proven.

Phase 1. The goal in phase one was to simplify the problem as much as possible but still demonstrate that the proposed approach is a feasible solution. Toward this end the architecture and behavior of the WSN was simplified. The number of nodes was limited to less than 10 and those nodes were specifically located so that the optimal clustering configuration could be determined as detailed in Appendix B. It was also assumed that each node would only send one message. The ratio *e* was also be set to 0 to simplify the fitness function.

The clustering method was also simplified in phase 1. The algorithm was configured to require each *CH* to be within direct transmission range of another CH. This resulted in a 1-radius constraint; restricting the radius of the clusters to the transmission range of the nodes. This constraint removed the additional complexity required to include gateway nodes in the network and also reduced the complexity of the fitness function.

The genetic algorithm was also simplified for phase 1. As noted above the chromosomes were configured so that each gene represented exactly one node in the network. Each gene had a value of either zero or one, with one indicating the node was a *CH*. The selection algorithm was limited to a simple elite selection. The chromosomes were ordered according to their fitness function score. The top-scoring half of the chromosomes was not modified so they carried over into the next generation. Crossover was not implemented during this phase. The top half was duplicated and one gene from each chromosome was randomly selected and mutated in order to create the remainder of the next generation.

The fitness function used for scoring the chromosomes was reduced in complexity by the simplifications already listed. These simplifications reduced the energy function (EQ 5) so that all the values were constants other than the number of messages sent and received at each node.

$$f(E_{recv}, CH) = \sum_{i=1}^{n}[x_i * k + y_i] * E_{recv} + e|CH| \quad (5)$$

Setting the constants to a value of one yielded equation (14) below:

$$f(E_{recv}, CH) = \sum_{i=1}^{n}[x_i + y_i] + |CH| \quad (14)$$

Therefore in phase 1 the fitness function consisted of calculating the messages sent and received at each node. In order to find the path for each message a pathfinding algorithm was implemented. The pathfinding algorithm was modified to return a set of nodes instead of a Euclidean distance. Since the goal was to minimize the target function the lowest fitness score was the optimal score. The chromosomes were sorted from lowest score to highest and the top half were used for creating the next generation as detailed above.

In this phase the location of the nodes was recorded in a Microsoft Excel document and then loaded into a C# application from the document so that the same number of nodes and their

respective locations were evaluated with each run of the application. As the node information was loaded the number of nodes was recorded and this was the number of genes used for each chromosome. During initialization each gene was assigned a random value of 0 or 1. The chromosomes were then evaluated using the simplified fitness function described above. Network configurations found to not provide for communication of all nodes to the sink were assigned a maximum (worst performance) value and not evaluated through the fitness function. The top performing half, those with the lowest fitness value, were retained into the next generation and had mutation applied at 100% probability in order to create the second half of the next generation. This process was iterated for a predetermined number of cycles and the results evaluated. A simplified representation of the process is shown below.

1. Load node information and save number of nodes
2. Create Chromosomes with same number of genes as number of nodes
3. Evaluate chromosomes and calculate fitness score for each
4. Retain top performing half and mutate to repopulate
5. Iterate steps 3 and 4 predetermined number of times
6. Evaluate results

This process provided a good baseline determination if the algorithm was successful for a simplified network and fitness function. The completed algorithm was tested on several sets of nodes. As noted above, the number and position of the nodes was specified so that the optimal clustering could be determined as detailed in Appendix B. The results from the algorithm were compared to the optimal in order to determine the level of success of the algorithm. It was found

at this stage that the algorithm was able match the optimal cluster configurations. The algorithm was modified and retested when it didn't match the optimal. Once the algorithm was determined to be successful work proceeded to Phase 2.

Phase 2. Phase 2 focused on improving the genetic algorithm (GA). It was expected that in order to see a significant difference in performance of the GA, the complexity of the WSN being clustered had to increase. Therefore the first step of Phase 2 was to increase the network complexity and test the algorithm from Phase1 with several different, more complex configurations. Next the various selection schemes used to choose the parents for the subsequent generation of chromosomes were tested. The schemes tested included Elite, Roulette-Wheel, Rank, and Tournament (Bayrakli & Erdogan, 2012) as from the literature these appeared promising for this problem. Performance of the GA in Phase 2 was measured based on the number of generations required to obtain an optimal clustering solution.

Crossover was another typical feature that was tested during Phase 2. Crossover is used to combine a set of parent chromosomes into two new offspring for the next generation. A random gene is chosen and all of the genes after that one are swapped between the parents. Given the constraints on the chromosomes as far as suitability for a WSN, it was unknown if crossover would improve or degrade the performance of the GA. Therefore the simple crossover just described was tested first.

The next steps in attempting to improve the performance of the GA were related to mutation and gene repair. Although a simple mutation was included in Phase 1, Phase 2 included a probability factor to indicate if mutation occurs or not. Different levels of probability were tested to determine the optimal level of mutation. Similar to mutation, chromosome repair

was tested as well at this stage. Repair involved modifying the chromosome prior to calculating its fitness score. The modifications changed genes to create a suitable number of cluster heads or other changes as applicable to make the chromosome able to provide communication from all nodes of the WSN to the sink.

The algorithm execution in Phase 2 followed a similar process as Phase 1. The nodes were recorded in and loaded from a Microsoft Excel document. The number of nodes and complexity of layout increased significantly from Phase 1. There were four additional areas where the complexity of the algorithm was increased from Phase 1. These were the probability-based mutation, crossover, chromosome repair, and multiple selection methods.

Additional selection criteria were tested in Phase 2. Elite selection was performed as noted in Phase 1. The top-performing half of the chromosomes were retained into the next generation. However in Phase 2 instead of mutation, crossover was used to generate the remaining members of the next generation. Two parents were chosen at random from the top-performing half and crossover was implemented as described above with the crossover point chosen at random. This was repeated until the next generation of chromosomes was complete; meaning the number of chromosomes in the original population of chromosomes was reached.

Tournament selection was also tested. In this phase Binary tournament selection was implemented. Two chromosomes were selected at random and their fitness scores compared. The winning chromosome, the one with the better (lower) fitness value, was retained as parent. Two more chromosomes were selected at random with the winner from those two selected as the second parent. Crossover was implemented on the parents in order to generate new

chromosomes. This process was repeated until the next generation of chromosomes was complete.

In the Linear Rank selection method the chromosomes were assigned a rank based on their fitness value. The chromosomes were sorted from most preferable to least preferable (lowest to highest) and then a probability was assigned according to equation (15) below:

$$P_i = \frac{numC - i}{(numC \times (numC + 1))/2} + P_{i-1} \tag{15}$$

In equation (15), P_i is the probability assigned to the current chromosome, and *numC* is the total number of chromosomes. After the probability was assigned to all chromosomes two chromosomes were selected based on probability of selection. Those two selected chromosomes were used as parents with crossover then applied to them to create two new chromosomes for the next generation. This process was repeated until the next generation of chromosomes was complete.

Roulette-Wheel selection had to be modified to be used in this work because this was a minimization problem and Roulette-Wheel was designed for maximization problems. This is because in Roulette-Wheel, the fitness score of each chromosome is divided by the sum of the fitness scores and this value is used to determine the probability of selection of each chromosome for the next generation. However, since a lower score was preferable in this work, the standard method was not feasible. Also the potential fix of subtracting each fitness value from the maximum fitness value would not work in this situation. Within each generation there were multiple chromosomes with the maximum score because as noted above the maximum fitness value (least preferable) was assigned to all chromosomes that did not result in a complete network. Subtracting each fitness score from the maximum would result in many chromosomes

having a fitness score of zero and therefore zero probability of being selected. In order to remedy this, each fitness score was subtracted from the sum of the maximum and minimum fitness scores. This is shown below in equation (16) where *FS* is the fitness value.

$$FS'_i = FS_{max} + FS_{min} - FS_i \tag{16}$$

The next step was similar to the Linear Rank method where the chromosomes were sorted based on the fitness value calculated with equation (16) from best to worst. Then a probability was assigned using equation (17).

$$P_i = \frac{FS_i}{\sum_{j=0}^{numC} FS_j} + P_{i-1} \tag{17}$$

In equation (17) P_i is the probability assigned to the current chromosome, *FS* is fitness value and *numC* is the number of chromosomes. After the probability was assigned to all chromosomes two chromosomes were selected based on probability of selection. Those two selected chromosomes were used as parents with crossover then applied to them to create two new chromosomes for the next generation. This process was repeated until the next generation of chromosomes was complete.

Mutation was performed in the same manner as in Phase 1 except that it did not always occur. A probability of mutation was predetermined. As every new chromosome was created based on a selection method detailed above it was determined if mutation occurred based on that probability. If mutation occurred it was done as described above with one gene randomly selected and the value of that gene changed.

The final addition in this phase was chromosome repair. Multiple repair methods were tested during this phase. The simplest was to randomly add cluster heads to the chromosome.

The criteria for deciding to add additional cluster heads was based on comparing the newly created chromosome to the most successful chromosome from the previous iteration. If the new chromosome had fewer cluster heads than the previous best configuration, then cluster heads were added. This could not be applied to all chromosomes since successive generations should create network configurations with fewer cluster heads. Therefore the criterion was tested using a percentage of the number of cluster heads from the previous generation. The algorithm determines the number of cluster heads from the best chromosome of the previous generation and any new chromosomes that have fewer cluster heads than a percentage of that number have cluster heads added at random locations. Different percentages were tested.

For the description of the other repair methods to be tested, recall Figure 9.

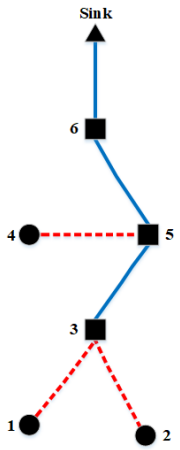

Figure 9. 6-Node WSN

The next method of repair tested was to repair the red (dashed) graph as shown above. Each node was checked to determine if it was within range of a cluster head or the sink. If it was not, then that node was made a cluster head and process continued for the remaining nodes with the new cluster head now being considered. This continued until all the nodes had been tested.

The criterion to check if this repair algorithm was applied to a chromosome was the same as the repair method above.

Another repair method tested in this phase was repair both the *RED* (dashed) and *BLUE* (solid) graphs shown above and also remove unnecessary cluster heads. First the *RED* (dashed) graph was tested to determine if all nodes were within range of a cluster head or the sink. If not then a cluster head was added but instead of randomly, the cluster head was added based on the shortest path of the nodes of the WSN. A shortest-path tree was created for each node in the network. The node that was in the shortest path for the most nodes that was not already a cluster head was made a cluster head. The check against the *RED* (dashed) graph was then run again. This process was repeated until the *RED* (dashed) graph was feasible; meaning all nodes were within transmission range of a cluster head or sink.

Next the *BLUE* (solid) graph was tested. Each cluster head was tested to see if it had a path through the *BLUE* (solid) overlay network to reach a sink. If this check failed, then a cluster head was added in the same manner as above, converting the node most used in shortest-path trees to a cluster head. After the new cluster head was added the test of the *BLUE* (solid) graph was executed again. This process was continued until the *BLUE* overlay graph was feasible.

The final part of this repair method was to remove unnecessary cluster heads. In later phases having fewer cluster head became more important since they incurred a greater energy cost than member nodes. Similar to the above processes, a shortest-path tree was constructed for every node in the network. The cluster head that was used the least number of times in the shortest-path trees was converted to a member node. Then the *RED* graph check and the *BLUE*

graph check were run again. If both passed then another cluster head was converted to a member node. This process was continued until one of the graph tests failed. In that case the last node converted to a member node was converted back to a cluster head.

The performance measurement in Phase 2 was based on a combination of quality of the solution based on the fewest number of cluster heads required and the number of generations required to obtain an optimal clustering configuration. However it was noted in early testing that a potential issue with this approach was a large number of chromosomes were required to initially have at least few that resulted in a suitable network. It was expected that chromosome repair may reduce the number of chromosomes required and therefore this was included in the performance measurement.

Phase 3. The next phase added complexity into the fitness function. The intent for this phase was to simulate a more realistic network messaging scenario. The variables for number of messages sent(x) and received(y) as well as the ratio of *Erecv* to *Esend* (k) and a value for *Erecv* were added back into the fitness function. Different random values within a fixed range were applied to each node and used for the number of messages sent. The number of messages received at each node were calculated based on the number sent from downstream nodes. A range of realistic values from the literature were substituted for k and *Erecv*. The value e that indicated the extra energy cost for cluster head maintenance was included in the fitness function. This required significant restructuring of the algorithm used for calculating chromosome fitness.

The different selection methods implemented in Phase 2 were continued in this phase with all of them tested again using the more complex fitness function. The repair method deemed the most successful from the previous phase continued to be used as well. As noted

previously, in this phase configurations that provided a feasible network with fewer cluster heads were preferable due to the increased energy usage of cluster heads.

The work in this phase began by generating a random number of messages to be sent from each node and each message included a randomly generated label that indicated which round it occurs in. The number of messages was limited from 1 to 10, as well as the number of rounds. These values were recorded in the same Excel document that contained the node information so that a consistent number of messages and occurrence schedule was maintained for testing. The number of messages received at each node was calculated based on the messages sent through that node to a sink. Messages from member nodes to their respective cluster head were not aggregated at the cluster heads. In order to simulate more realistic conditions, it was assumed multiple messages from the same member node were not be sent simultaneously. Therefore, the energy cost of sending and receiving messages was calculated using rounds. As noted, each message that was assigned to a node also had a randomly generated label indicating which round it occured in. For example, in round one only nodes that had a message indicated in round one sent a message.

Given that sensor nodes in typical applications will send messages frequently, the probability used to determine if a node sends a message in any round was set initially at 75%. This probability was set to different values using the same number of nodes and node locations in order to test if there was a significant effect on the performance of the algorithm.

Testing for early parts of this phase consisted of comparing the results of the algorithm to optimal configurations. The algorithm was tested with each selection method and with the optimal mutation rate and optimal repair method determined in previous phases. It was expected

at this stage that the GA would approximate the optimal configuration, otherwise the algorithm was modified and retested. If the algorithm didn't perform as expected, points of failure were determined through testing. Modifications of work from previous phases were then made, including investigating mutation probability, selection criteria, crossover, and repair method. Changes were made and tests from Phase 3 were run again. Once the algorithms in this phase performed at an acceptable level, work proceeded to Phase 4.

Phase 4. Phase 4 expanded the 1-radius constraint on the distance between cluster heads to 2-radius by incorporating gateway nodes into the overlay network. This did not effect the 1-radius constraint on the size of clusters. Intuitively the addition of gateway nodes should have reduced the number of cluster heads required to create a suitable network.

This can be seen by considering a version of the network shown in Figure 10, modified to include gateway nodes.

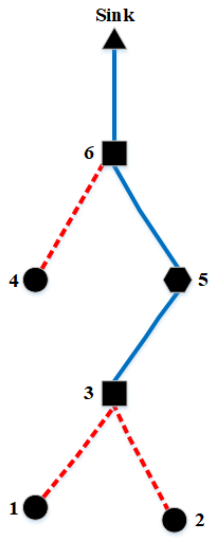

Figure 10. WSN with gateway node

As shown above, the gateway node depicted as a hexagon (node 5), reduces the required number of cluster heads without reducing the connectivity of the network. Because gateway nodes only had to maintain routing information for one node, the same as member nodes, and messages were not aggregated at gateway nodes, they did not incur additional energy costs as compared to member nodes.

Adding gateway nodes to the network required multiple changes to the algorithm. At initialization genes were assigned a value of "0" indicating member node, "1" indicating cluster head, or "2" indicating gateway node. The fitness function was modified to include gateway nodes in the *BLUE* graph to determine if a network was complete. Also, no *RED* graphs were permitted to be connected to a gateway node for purposes of determining if the network configuration provided for communication from all nodes.

The repair methods also required modification. When changing member nodes to create a complete network, it had to be determined if the node should become a cluster head or a gateway node. Repair processes that converted nodes back to member nodes had to be modified in order to determine if it was a cluster head or gateway node that should be converted.

Mutation was also modified. Instead of simply changing the value from "0" to "1" or vice-versa, mutation had to use a random number to determine what value a gene should be changed to. Member nodes were mutated to either a cluster head or gateway node with an even probability of each. Similarly cluster heads were mutated to member nodes or gateway nodes, and gateway nodes were mutated to member nodes or cluster heads.

Testing and measurement for this phase consisted of comparing the results of the algorithm against the same network both with and without gateway nodes. Performance

measurement was based on the fitness value. Given the increases in complexity of both network architecture and the fitness function in phases 3 and 4, the GA methods for selection, crossover, mutation, and repair were reconsidered during this phase to ensure optimal methods were still being employed.

Phase 5. The first step in Phase 5 was to add in energy awareness to the cluster selection method. Rather than modifying the target function, the energy-aware system was added separately to the fitness function. Each send or receive operation added the appropriate amount of energy to energy consumed by a node based on the pre-determined values for E_{send} and E_{recv}. The number of messages sent for each node was random within a range as was implemented in Phase 4.

Crossover, mutation, and repair operations were modified in order to add the energy aware system along with a new variable for each node to indicate its energy usage. As the rounds were run within the fitness function the energy usage at each node was increased in accordance with the predetermined values for E_{send} and E_{recv}. Cluster heads had their energy usage increased additionally to account for the overhead of maintaining routing information.

The results of this phase were evaluated based on which algorithm could create a network where the energy usage the most popular node; the node involved the most in messaging, had the least energy usage. This was the last phase of planned work for this research. At the end of this phase, an algorithm was developed to improve the lifetime of a wireless sensor network across a range of environments.

Data set

The data used to test the proposed work was from different sources depending on the phase. Early work required verification of success as compared to optimal configurations and therefore the network architecture was deliberately kept simple. Placement of the nodes was predetermined and the nodes were positioned so that the determination of the optimal network configuration was possible as detailed in Appendix B. Additional data sets of nodes and locations for later phases were developed from a method written in C#. This was necessary since existing p-median test data sets were not constrained in the manner described in the problem statement. This is not unusual since other implementations of GAs for WSNs in the literature have used their own simulation data (Correa et al., 2004; Bayrakli & Erdogan, 2012). The C# program was written to randomly place nodes within a fixed, square area. In order to represent more realistic conditions a threshold restricted how near neighboring nodes could be placed.

Resources

The algorithms were written and tested on a PC running a current generation Intel i7 6700HQ processor and 16GBs of RAM. The PC also had an nVidia GeForce GTX 960M discrete graphics processor (GPU) so that the machine learning code could take advantage of the parallel processing power of the GPU. The primary language for creating test data sets and algorithm implementation was C#, therefore the C# IDE Microsoft Visual Studio 2017was used. The operating system was Microsoft Windows 10.

Summary

As shown above, the work for this dissertation proceeded in stages. It began with simple node positioning and a limited number of nodes and then each phase increased in complexity. Different approaches were implemented and tested at each phase. Work on the subsequent

phases did not begin until the previous phase had been tested and shown to be successful by the measure noted for each phase. Proceeding in this incremental manner facilitated steady progress and revealed flaws in the approach early on so that significant rework was not required in later stages.

Results

Overview

This dissertation explored the use of a genetic algorithm for clustering the nodes of a wireless sensor network. Chromosome repair methods were developed and tested. The successful methods were tested on multiple networks and the results are presented below divided into phases similar to those outlined in the specific methods section.

Phase 1. The intent of Phase 1 was to simplify the initial problem statement so that a smaller scale experiment could be set up and used to test the feasibility of the proposed idea. Several modifications were made to reduce the overall complexity of the problem. First the arrangement of the nodes was simplified from the random positioning of wireless sensor nodes typically seen in the literature. Recall that the nodes of the wireless sensor network (WSN) were considered vertices on an undirected graph $W(V \cup S, E)$ where $\mathrm{V} = \{v_i \mid 1 \leq i \leq n\}$ represented the sensor nodes were and $S = \{s_i | 1 \leq i \leq c, s_i \notin V\}$ were the sinks. Nodes for the case study were placed in fixed positions. The network was configured to have only one sink and the number of nodes was also reduced to further simplify the problem. This simplification served multiple purposes. The simplification made it possible to determine the optimal network configuration so that the results obtained through the algorithm could be compared to the optimal configuration. The methods used to determine the optimal configurations are detailed in Appendix B. The simple network also provided a baseline for comparison of later experiments.

The number and locations of the nodes and sink were recorded as a two-dimensional table showing *x* and *y* coordinates. This paradigm would be continued throughout this work so that algorithms could be consistently compared to the same environment. A sample table depicting

the 6-node network used in early testing is shown below in Table 1. The ID of the sink node was negative so it could be easily differentiated during calculations.

ID	xCoord	yCoord
1	0	1
2	0	2
3	0	3
4	0	4
5	1	3
6	1	1
-1	0	4.8

Table 1. 6-node network

The target function detailed in the problem statement was modified to reduce complexity. The original target function (eq. 5) included terms for the number of messages being sent and received. For Phase 1 the target function was reduced by assuming each node would only send one message. It could not be assumed that each node would only receive one message since cluster heads would receive messages from all of their member nodes as well as messages from other cluster heads.

$$f(E_{recv}, CH) = \sum_{i=1}^{n}[x_i * k + y_i] * E_{recv} + e|CH| \quad (5)$$

Recall that in equation (5) the variables x_i and y_i represented the number of messages being sent and received at node v_i respectively. The constant k was a factor representing the extra energy required to send versus receive transmissions as defined in (Wu et al., 2002).

$E_{send} = kE_{recv}$, $k \geq 1$ (Wu et al., 2002)

The constant e accounted for the extra energy consumed by cluster heads versus member nodes. It represented the energy required for the average cluster head to maintain routing information and actively listen for messages from member nodes. Sinks are typically connected

to a substantial power source and were therefore energy usage at sinks was not considered in the problem to minimize the energy usage of the network.

The receive energy for each node along with the constant *k* were also set to one for the initial phase of the case study and the constant *e* was set to one. Therefore the target function was reduced to the equation:

$$f(E_{recv}, CH) = \sum_{i=1}^{n}[x_i + y_i] + |CH| \tag{14}$$

The complexity of the original problem was also reduced through not considering gateway nodes and enforcing the 1-radius constraint. The 1-radius constraint dictates that the radius of a cluster can be no larger than the transmission range of the nodes. Another way to express this is that all member nodes must be one hop from a cluster head. It also means that all cluster heads must be within transmission range or one hop of another cluster head. Recall that distance within the wireless sensor network was defined as three-dimensional Euclidean distance. Within the graph W distance between two nodes $q \in V$ and $p \in V$ was therefore given by:

$$d(q,p) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2 + (q_z - p_z)^2} \tag{2}$$

Using the definition for distance above, the edges of the graph W were defined using the function $MA_{connect}$ where R represented the transmission range of the nodes:

$$MA_{connect}(v_i, v_j) = \begin{cases} 1\ ;\ d(v_i, v_j) \leq R \\ 0\ ;\ d(v_i, v_j) > R \end{cases} \tag{1}$$

Therefore in Phase 1given the set of cluster heads $CH = \{ch_i | 1 \leq i \ \leq m, ch_i \in V\}$, sinks $S = \{s_i | 1 \leq i \ \leq c, s_i \notin V\}$, and member nodes $B = \{b_i | 1 \leq i \ \leq n, b_i \in V, b_i \notin CH\}$ the WSN satisfied the 1-radius constraint iff:

$$\forall\, b \in B: \sum_{i=1}^{m} MA_{connect}(CH_i, b) \geq 1 \ \ AND$$

$$\forall\, ch \in CH: \{\exists ch_i, ch_i \neq ch: MA_{connect}(ch_i, ch) = 1 \ OR \ \exists s_i: MA_{connect}(s_i, ch) = 1\ \}$$

The chromosome encoding was done such that each gene represented a node and each chromosome included exactly the number of genes so that all of the nodes in the wireless sensor network (WSN) were represented. Each gene was a binary variable with a value of one indicating that a gene was a cluster head. Therefore each chromosome was represented by a $1 \times n$ binary matrix:

$$MA_{chromosome} = [g_1, g_2, g_3, \dots g_n].$$

The encoding of the genes indicated which nodes were members of *CH* and which were members of *B* according to equation (11) below.

$$g_i = \begin{cases} 1\ ;\ v_i \in CH \\ 0\ ;\ v_i \in B \end{cases} \tag{11}$$

The GA was initialized with a predetermined number of chromosomes. The variable CH_{prob} was defined as the probability that a node would be selected initially as a cluster head and was applied to all nodes in the WSN. The value of each gene of the chromosome was assigned either zero or one based on CH_{prob}. CH_{prob} was set to 50% for Phase 1 since preliminary testing showed the optimal configuration of small networks contained approximately 50% cluster heads. This was due to the sparse nature of the test networks. Each node in the small test networks typically had only one or two other nodes within transmission range, therefore about half of the

nodes had to be cluster heads in order to make the networks feasible. This could be seen later in Figures 15 and 16. Next the target function was run against all of the chromosomes to determine their fitness value.

The target function was the same as equation (6), previously detailed in the specific methods section with the energy function reduced as shown above.

$$TF() = \begin{cases} f(CH)\ ; & if\ RED(B) = true\ AND\ BLUE(CH) = true \\ \infty & ;\ otherwise \end{cases} \tag{6}$$

Recall the RED() and BLUE() functions that were used to determine network feasibility. The RED() function tested if all member nodes could connect to a cluster head and the BLUE() function tested if every cluster head was connected to a sink.

$$RED(B) = \begin{cases} true\ ; & if\ \forall\ b \in B: \sum_{i=1}^{m} MA_{connect}(CH_i, b) \geq 1 \\ false\ ; & otherwise \end{cases} \tag{3}$$

$$BLUE(CH) = \begin{cases} true\ ; & if\ \forall\ ch \in CH: ch \in S_{comp} \\ false\ ; & otherwise \end{cases} \tag{4}$$

S_{comp} represented the set of nodes that formed the connected component of the sink(s).

Several steps were involved in the fitness function. First the network configuration (the selection of cluster heads and member nodes) defined by the chromosome was evaluated against the RED() function. If it returned false, the chromosome was assigned the least preferable fitness value. Since this was a minimization problem the least preferable value was the maximum value.

For the next step in the fitness function the Breadth-First Search (BFS) algorithm was implemented and run against *CH* with the sink as the root node. The algorithm is shown below in Figure 11.

```
BFSwsn
Inputs:
sink node
N = {set of network nodes to traverse}
Outputs:
closed = {set of nodes traversed by algorithm}

1 open = {empty set}
2 closed = {empty set}
3 add sink to open
4 while(open ≠ Ø)
5 {
6       ne = first node in open
7       remove all children from ne
8       for(each n in N)
9       {
10          if(MAconnect(ne,n)AND(n not in { open, closed }))
11          {
12                add n to children of ne
13                add n to open
14          }
15      }
16      remove ne from open
17      add ne to closed
18 }
19 return closed
```

Figure 11. Algorithm BFSwsn

BFS works through starting at a single node. In this case the starting node was the sink as shown by adding the sink to the set of open nodes in line 3. The first node in the open set was set as the node currently being evaluated (*ne*) as shown in line 6. The node *ne* was evaluated by adding its neighbors, line 13, to the open set. Afterward *ne* was added to the closed set and removed from the open set. The process was repeated until the open set was empty. BFS for WSN (BFS_{wsn}) was implemented with the primary difference from standard BFS being that nodes were considered neighbors or not of ne based on the result of $MA_{connect}$ for the nodes as shown in line 10. Also in BFS_{wsn} the neighbors that each *ne* added to the open set were

considered children of *ne* as shown in line 12. Each node's children were used later to compute the path to the sink which was useful in calculating the fitness value.

This accomplished two goals, the first was to determine if the overlay network formed by the cluster heads facilitated communication from all cluster heads to the sink: i.e. the BLUE() subroutine was true. BLUE() returned true if and only if at the end of running BFS_{wsn}, all cluster heads were in the closed set. If the BFS_{wsn} algorithm couldn't resolve a path from the sink to all cluster heads then the chromosome was assigned the worst fitness value. The second feature of using BFS_{wsn} was that the children of each node could be recorded and used later in fitness value calculations.

As shown above in the reduced target function, the only terms that weren't constants were x_i, y_i, and $|CH|$. Therefore the factors in determining the fitness value were message hops and the number of cluster heads. In the final step of the fitness calculation, each member node was assigned to the cluster of its nearest cluster head based on the Euclidean distance as shown in equation (2). With cluster assignment and the number of hops from each cluster head to the sink, the target function could be calculated for the chromosome.

The code for creating and running the genetic algorithm (GA) was written in C#. The genetic algorithm process at this phase is shown below in Figure 12.

Genetic Algorithm

1. Load node information
2. Create Chromosomes with same number of genes as number of nodes
3. Assign fitness value; if RED() or BLUE() return false; maximum value, otherwise calculate fitness value
4. Retain top performing 20% and mutate to repopulate chromosomes
5. Iterate steps 3 and 4 predetermined number of times

Figure 12. Genetic Algorithm

In early testing many of the chromosomes did not satisfy the criteria required for the network; either each member node was not in range of at least one cluster head or all of the cluster heads were not within range of at least one other cluster head. In order to ensure that the chromosomes that provided a feasible network were carried over to the next generation, elitism was used for selection of the next generation of chromosomes. In the first iteration only one or two chromosomes were typically feasible networks. In later iterations most or all chromosomes were feasible networks. Therefore the Elite selection method was configured to retain the top 20% for the next generation so that in each generation chromosomes were retained that represented feasible networks, but there was still room for new chromosomes that resulted from mutation. Mutation was performed by selecting a gene at random and its value changed to its complement. Next the fitness score was calculated for all chromosomes and the process repeated for a preset number of iterations.

As noted above the algorithm was run against simple network configurations so that the results could be easily verified against optimal configurations. The two network configurations tested are shown below in Figures 13 and 14.

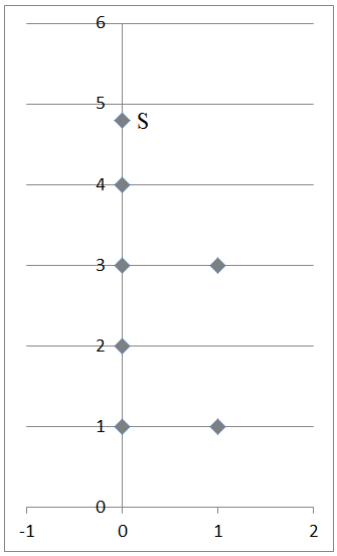

Figure 13. 6-Node Network

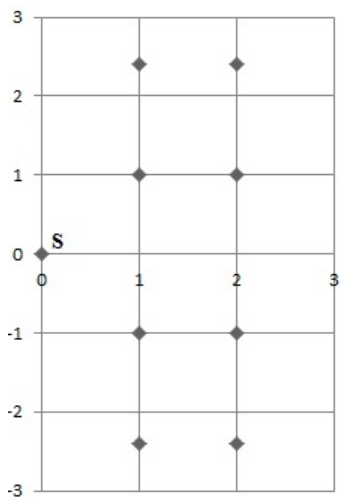

Figure 14. 8-Node Network

As shown configuration 14 consisted of six nodes indicated by diamonds and a sink indicated by a diamond with an “S” label. The GA was run with 20 chromosomes each with 6 genes and the transmission range set to 1.5 units. This meant that the cluster radius and distance between cluster heads was limited to 1.5 units as well. Typically within 2 or 3 iterations an optimal solution was found. An optimal configuration is shown below in Figure 15.

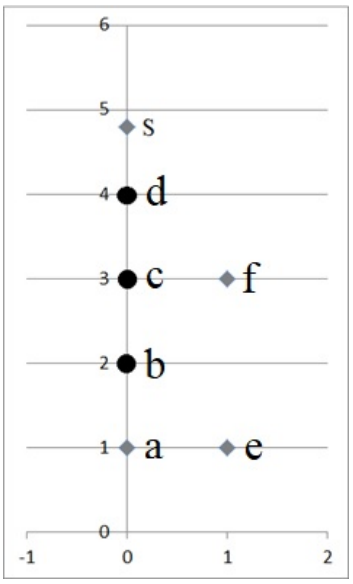

Figure 15. Optimal Configuration

The darker circles indicate the nodes that were selected by the GA as cluster heads (*b*,*c*,*d*). Nodes *a* and *e* were members of node *b* and node *f* was a member of node *c*. Also the overlay network can be clearly seen going through nodes *b*, *c*, and *d*, and then to the sink. The configuration in Figure 15 is optimal as it results in the lowest fitness value. This is because it creates a feasible network with the fewest number of cluster heads. For clarity the calculation of the fitness value is shown below. Recall the reduced target function:

$$f(CH) = \sum_{i=1}^{n-p}[x_i + y_i] + |CH| \tag{14}$$

Beginning at node *a* and proceeding alphabetically through the nodes yields:

$$f(CH) = [x_a + y_a] + [x_b + y_b] + [x_c + y_c] + [x_d + y_d] + [x_e + y_e] + \left[x_f + y_f\right] + |CH|$$

$$= [1 + 0] + [3 + 2] + [5 + 4] + [6 + 5] + [1 + 0] + [1 + 0] + 3 = 31$$

It was noted during the testing of the 6-node network that the fitness value would be improved if instead of becoming a member of the nearest cluster head, node *f* was a member of node *d*. In this case the fitness value would be:

$$f(CH) = [1 + 0] + [3 + 2] + [4 + 3] + [6 + 5] + [1 + 0] + [1 + 0] + 3 = 29$$

As shown the fitness value is lower and therefore preferable. However at this stage of testing, member nodes would continue to be members of the nearest cluster because it was not clear that this approach would be preferable in reducing the energy usage of complex networks. It may be preferable to balance the membership of clusters; therefore this question was explored further in later phases.

Figure 16 below shows one of several possible optimal configurations for the 8-node WSN shown in Figure 14.

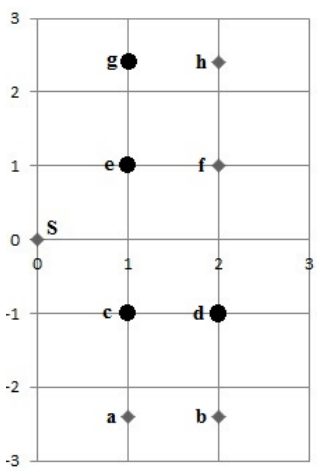

Figure 16. Optimal Configuration

Again 20 chromosomes were used but with 8 genes each and a transmission range of 1.5 units. An optimal solution containing 4 cluster heads was typically found by the GA within 3 or 4 iterations. The cluster heads selected by the GA are shown as the darker circles. As shown

there are two overlay paths to the sink; through nodes *d* and *c* and through nodes *g* and *e*. This configuration satisfies the constraints above with each member node a member of the nearest cluster head.

As shown by the diagrams, the GA was able to find an optimal solution for each WSN. While these were simple example networks, they demonstrated the feasibility of using a GA in order to determine an optimal network configuration.

Phase 2. The first change for Phase 2 was to test the GA on a more complex network configuration. The new network consisted of 44 nodes and a sink. This more complex WSN was used to confirm earlier results on a complex network and also to test additional GA options including different selection methods, crossover, and mutation based on probability. The layout of the 44-node WSN is shown in Figure 17.

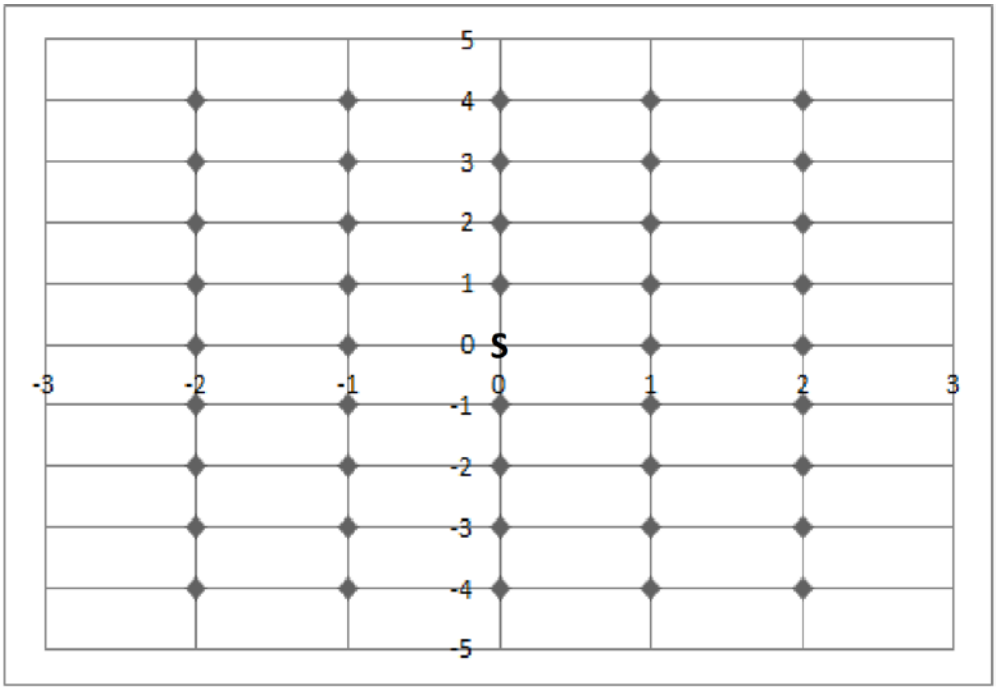

Figure 17. 44-Node Network

In Figure 17 the diamonds represent the nodes of the WSN and the sink is shown at the origin represented by an “S”.

Recall that during Phase 1, Elite selection was used exclusively with no crossover and a fixed mutation rate. In Phase 2, additional selection methods were introduced along with the use of crossover. Crossover was implemented using a random number for the crossover position and a probability was used to determine if crossover occurs. The two parent chromosomes were selected using one of the selection methods. Once the parents were selected, crossover was implemented to create the next generation of chromosomes. A crossover point *cp* was selected randomly where $1 \leq cp \leq n$ and *n* was the number of genes Given two parent chromosomes *p* and *q*, crossover created two new chromosomes *x* and *y* using the equations below.

$$x_i = \begin{cases} p_i & i \leq cp \\ q_i & i > cp \end{cases}$$

$$y_i = \begin{cases} q_i & i \leq cp \\ p_i & i > cp \end{cases}$$

In the equations above, x_i represented the gene at position *i* in chromosome *x*.

Given two chromosomes, C1 and C2 below, assume the randomly selected crossover point is the third gene.

C1:	0	0	0	0	0	0
C2:	1	1	1	1	1	1

The two new chromosomes, NC1 and NC2, that resulted after crossover are shown below.

NC1:	0	0	0	1	1	1
NC2:	1	1	1	0	0	0

The Elite selection scheme used in Phase 1 was tested on the 44-node network along with more complex selection schemes: Roulette-Wheel, Linear Rank, and Tournament selection.

The Roulette-Wheel selection method had to be modified for use with a minimizing optimization problem. This was because in Roulette-Wheel, the fitness score of each chromosome is divided by the sum of the fitness scores and this value is used to determine the probability of selection for the next generation. However, since a lower score was preferable in this work, the standard method was not feasible. Also the method of subtracting each fitness value from the maximum fitness value wouldn't work in this situation. Within each generation, there were multiple chromosomes with the maximum value because the maximum was assigned to all chromosomes that did not result in a feasible network. Subtracting each fitness value from the maximum would have resulted in many chromosomes having a fitness value of zero and therefore zero probability of being selected. In order to remedy this, each fitness value was subtracted from the sum of the maximum and minimum fitness values. In equation (16) *FS* is the fitness score.

$$FS'_i = FS_{max} + FS_{min} - FS_i \tag{16}$$

The probability was assigned using equation (17) where *numC* was the number of chromosomes. This was a slight modification of the formula used in Penchev, Atanassov, & Shannon (2009).

$$P_i = \frac{FS'_i}{\sum_{j=0}^{numC} FS'_j} + P_{i-1} \tag{17}$$

As shown below the results of the Roulette-Wheel selection method were less preferable as compared to the other methods. This was likely due to the large difference from minimum to maximum fitness value of three orders of magnitude. Further research may be done to determine

if there is a better method for modifying Roulette-Wheel selection or if it is simply not well suited for minimization problems with such a large variance in values.

In the Linear Rank selection method the chromosomes were assigned a rank based on their fitness score. The chromosomes were sorted from best to worst (lowest to highest) and then a probability was assigned according to equation (15) below:

$$P_i = \frac{numC - i}{(numC \times (numC + 1))/2} + P_{i-1} \tag{15}$$

In equation (15) P_i was the probability assigned to the current chromosome, and *numC* was the total number of chromosomes. After the probability was assigned to all chromosomes two chromosomes were selected as parents based on probability of selection.

Tournament was implemented using Binary tournament selection. Two chromosomes were selected at random and their fitness scores compared. The winning chromosome, the one with the better (lower) fitness value, was retained as a parent. Two more chromosomes were selected at random with the winner from those two selected as the second parent.

For the 44-node WSN, 50 chromosomes were used as the population and each chromosome contained 44 genes with each node of the WSN encoded to a gene in the chromosome. A transmission range of 1.5 units was used in all testing on this network and a fixed mutation rate of 5%. The probability-based mutation process is described below. All test configurations were run 10 times and an average of the 10 runs used as the result.

The results from testing the four different selection methods are shown below in Table 2. Additionally the estimated optimal value for the 44-node network is shown for comparison. The

estimation method is detailed in Appendix B. As shown, the values achieved were within 4% of the optimal value except for Roulette, which was explained above.

	Elite	Roulette	Rank	Tournament	Optimal
Fitness Value	200.7	214.7	203.1	201.7	196
Iterations	215.6	406.9	125.1	143.7	n/a

Table 2. Selection Methods Results

As noted earlier each simulation was run 10 times with the average score used for the graph. The optimal value for a test was the least fitness value that was achieved on any particular iteration. Since this was a minimization problem, lower scores were preferable. The iteration where the least fitness value was first obtained was also recorded. For example the output of running Elite selection on the 44-node network is shown below.

value	iterations
200	127
200	106
200	362
200	170
199	108
199	82
209	492
200	274
201	115
199	320
200.7	**215.6**

Table 3. Output

These results were obtained with 100 chromosomes running for 500 iterations. There are 10 sets of values representing the 10 runs of the algorithm. The value column represents the lowest value that was obtained by any chromosome during the run and the iterations column records the iteration where that lowest value first occurred. As shown 90% of the runs reached

their lowest value well before the 500 iteration stopping point and therefore, it could be said with confidence that the average value was the optimal that this algorithm could achieve under the given conditions.

As shown, the fitness values were similar with the exception of Roulette-Wheel which was explained above. In this round of testing, there was no clear superior selection method. Although Elite selection obtained a slightly better value, Linear Rank and Tournament were able to achieve their optimal value in fewer iterations.

As noted, the above tests were run using a fixed mutation rate of 5%. This means that after each new chromosome was created via crossover, there was a 5% chance that one of its genes would be mutated. Mutation was performed by first randomly selecting the mutation point $|(1 \leq mp \leq n)$. Then the chromosome was be mutated using the equation below where *x* represents a chromosome and x_i represents the gene at position *i* in *x*.

$$x_i = \begin{cases} x_i & i \neq mp \\ x_i \oplus 1 & i = mp \end{cases}$$

The next set of testing varied the mutation rate from 0% to 5%. Considering the minimal variance in selection methods shown above, only the Elite selection method was used for testing the mutation rates. The results are shown below in Figures 18 and 19.

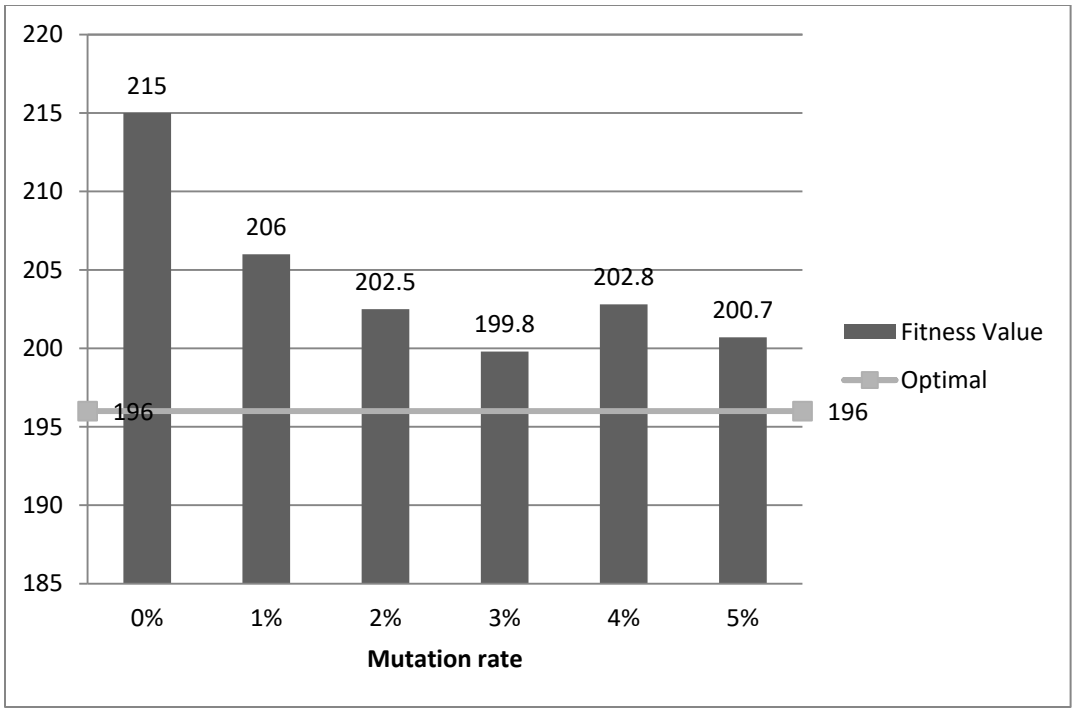

Figure 18. Mutation Results - Fitness

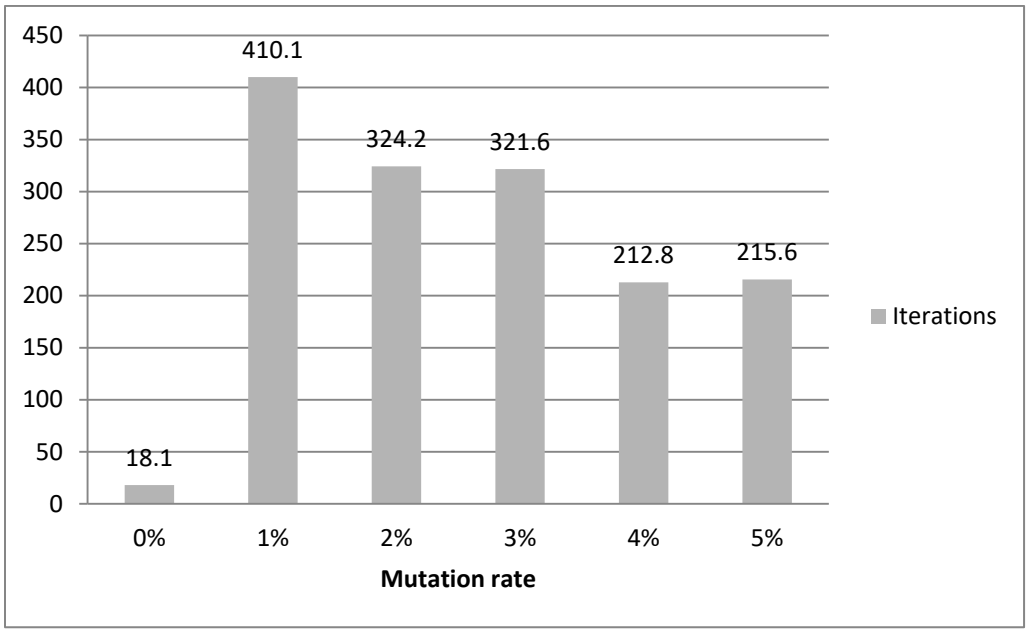

Figure 19. Mutation Results - Iterations

The results showed that there was minimal improvement in the fitness value with increasing the mutation rate once the rate was above 0% or no mutation. As shown the results

for the 3% and 5% mutation were within 2% and 2.4% of the estimated optimal fitness value respectively. This showed that the algorithm was performing well with the larger 44-node network. Also shown was that iterations required to reach the best values obtained were reduced significantly as the mutation rate increased. It was shown previously that the selection method did not significantly alter the fitness value other than the previously noted difficulty with roulette-wheel selection for this problem. Therefore in the remaining phases, testing was focused on the repair method as shown below and less on the testing of mutation rates and selection method.

The last part of Phase 2 involved developing and testing several methods of chromosome repair. The goal of chromosome repair was to provide more optimal solutions in fewer iterations. In this phase the genetic algorithm would be modified as shown below in Figure 20.

Genetic Algorithm

1. Load node information
2. Create Chromosomes with same number of genes as number of nodes
3. Assign fitness value; if RED or BLUE return false; maximum value, otherwise calculate fitness value
4. Use selection methods and crossover to create next generation
5. Run repair method
6. Perform mutation
7. Iterate steps 3 through 6 predetermined number of times

Figure 20. Genetic Algorithm

Repair method 1.

In the first method, the number of cluster heads in each new chromosome was compared to the number of cluster heads in the currently best ranking chromosome. This was used because

testing showed that typically the best ranking chromosome would represent a feasible network. The method is shown below in Figure 21.

```
Repair Add Random
Inputs:
C = {set of chromosomes }
repairLevel = 25%, 50%, 75%, or 95%

Outputs:
C = {set of chromosomes }

1 Cbest = chromosome with minimum fitness value from previous iteration
2 numCHbest = number of cluster heads in Cbest
3 for(each c in C)
4 {
5     if((number cluster heads in c)<repairLevel*numCHbest)
6     {
7           while((number cluster heads in c)< numCHbest)
8           {
9                 randomly add cluster head to c
10          }
11    }
12 }
13 return C
```

Figure 21. Add Random

The number of cluster heads in a chromosome was compared to the best ranking chromosome from the previous iteration as shown inline line 5 above. Chromosomes with fewer cluster heads had additional cluster heads added up to the number of the best ranking chromosome in order to help the chromosome provide a feasible network. The value could not be compared directly because ideally future generations would have fewer chromosomes in order to reduce the fitness value. Therefore the new chromosome's number of cluster heads was compared to a percentage of the current best chromosome's number of cluster heads as shown on the right-hand-side of the inequality in line 5. For example, using a percentage of 50%, if the current best chromosome contained 20 cluster heads, any chromosome with less than 10 cluster heads had 10 cluster heads added by changing the values of 10 genes in the chromosome from 0 to 1. The genes selected to convert to cluster heads were chosen at random as shown in line 9.

Chromosome repair was added to the other improvements and was tested with the 44-node network. Elite selection was implemented with crossover and 5% mutation probability. The Chromosome repair was tested at intervals of 25%, with 95% being used instead of 100% since 100% would have prevented the overall GA solution from improving. The results of using this repair method with the GA algorithm from Figure 20 are shown below in Table 4.

	25%	**50%**	**75%**	**95%**	**Optimal**
Fitness Value	205	203.9	202.4	205.3	196
Iterations	146.6	230.5	93.3	251.2	n/a

Table 4. Results Random Add

As shown in the table, generally chromosome repair was able to reduce the number of iterations required to reach the optimal solution. Unfortunately the fitness value was slightly less preferable. Therefor additional repair methods were attempted.

Repair method 2.

The second version of chromosome repair was done by evaluating each node that was not a cluster head. If the node was within range of a cluster head or sink then no change was made, otherwise the node was converted to a cluster head. While the previous method randomly added cluster heads up to a threshold, this version added cluster heads with the intent of making the network feasible. The method is shown below in Figure 22.

```
Repair Add CH
Inputs:
c = chromosome
Ouputs:
c = chromosome


B = {member nodes of c}
CHsi = {cluster heads of c and sink}
for(each b in B)
{
    for(each n in CHsi)
    {
          if(MAconnect(b,n))
          {
                continue(next b)
          }
    }
    convert b to a cluster head
}
return c
```

Figure 22. Add CH

Again this scheme was run with Elite selection with crossover and 5% mutation rate. Since this repair did not rely on the number of cluster heads it was run on a percentage of the lowest performing chromosomes of each iteration. For example when run at 20%, the chromosomes were sorted by fitness value and the least preferable 20% would have the repair method applied to them. The repair method was not run at greater than 80% of the lowest performing chromosomes, so that that the top 20% could remain for the subsequent generation. This method was run at 20% intervals with the results shown below in Table 5.

	20%	**40%**	**60%**	**80%**	**Optimal**
Fitness Value	204.7	200.4	204.1	202	196
Iterations	207.2	184.4	197	165.9	n/a

Table 5. Results Add CH

As shown, there was a slight general improvement over the previous repair method. The iteration count and fitness values were overall slightly better. Intuitively it seemed this method should have had significantly better results versus randomly adding cluster heads. In order to account for the lack of improvement, it was checked if mutation was interfering with the repair. However, when run with mutation turned off performance as far as the optimal fitness value was significantly degraded. Observing the values of the chromosomes while running the simulation, it was noticed that most of the chromosomes did not need repair based on the number of cluster heads. This was because during initialization, 50% of the genes were set as cluster heads. This was still a reasonable value because the optimal performing chromosomes on the 44-node network were approximately 40% cluster heads. This value would be adjusted in later work as networks became larger and required a lower percentage of cluster heads. In order to test this as the cause, the percentage of initial cluster heads was lowered to 10%, but this dramatically degraded overall performance of optimal fitness score. These results led to the next potential method of repair which was to evaluate methods for removing unnecessary cluster heads.

Repair method 3.

Based on the repair method results above, a third method of chromosome repair was implemented and tested. This method broke the repair process into three stages. First the RED() subroutine as defined above was run. If it returned false then the *RED* graph was repaired. Recall the *RED* graph consists of the member nodes, cluster heads, and the edges from member nodes to their cluster heads. Recalling that the graph of the WSN *W* was defined as:

$$W = (V \cup S, E)\ where\ E = \{(v, w) | MA_{Connect}(v, w) = 1\}$$

The *RED* graph was defined as:

$$RED = (B \cup CH, E_{RED})\ where\ E_{RED} \subset E\ and\ \forall\ (v, w) \in E_{RED}: v \in B\ and\ w \in CH$$

Repairing the *RED* graph ensured that all member nodes were within range of a cluster head. However unlike earlier versions, before adding a cluster head, this method first performed a Breadth-First Search for WSN (BFS_{wsn}) from the sink to all of the nodes of the network. This allowed a shortest-path to be determined for each node. The node that was not already a cluster head and that appeared most frequently in the shortest-paths was converted to a cluster head. The RED subroutine was then run again. This process was repeated until the *RED* graph check passed, indicating all nodes were within transmission range of a cluster head.

Next the *BLUE* graph was tested. Recall the *BLUE* graph consists of the cluster heads and the connected component that includes the sink.

$$BLUE = (CH \cup S, E_{BLUE})\ where\ E_{BLUE} \subset E$$

The BLUE() subroutine as defined earlier was run and if it returned false a cluster head was added in the same manner as above, converting the node most used in the shortest paths to a cluster head. After the new cluster head was added, the BLUE() subroutine was executed again. This process was continued until the *BLUE* overlay graph was feasible.

The final part of this repair method was to remove unnecessary cluster heads. Similar to the above processes, the shortest paths created during the BFS_{wsn} were used. The cluster head that was the least frequent in the shortest paths was converted to a member node. Then the RED() and BLUE() subroutines were run again. If both passed then another cluster head was converted to a member node following the same process. This process was repeated until one of

the subroutines failed. In that case the last cluster head converted to a member node was converted back to a cluster head.

A $1 \times n$ matrix $MA_{frequency}$ was created to store the frequency that each node appeared in a shortest path result set. Equation (18) below uses Iverson notation:

$$[P] = \begin{cases} 1 & if\ P\ is\ true \\ 0 & otherwise \end{cases}$$

to show how $MA_{frequency}$ is populated.

$$MA_{frequency}(v_j) = \sum_i^n [v_j \in BFS(v_{i,}s)] \tag{18}$$

Using $MA_{frequency}$ the function MAXFREQ() that returns the node with the maximum frequency value from $MA_{frequency}$ can be defined as:

$$MAXFREQ() = max_{v \notin CH}(MA_{frequency}(v_i), MA_{frequency}(v_j), \dots MA_{frequency}(v_n))$$

As shown *MAXFREQ*() returns the value of node with the greatest frequency that is not a cluster head. Similarly, the function *MINFREQ*() is defined below.

$$MINFREQ() = min_{v \in CH}(MA_{frequency}(v_i), MA_{frequency}(v_j), \dots MA_{frequency}(v_n))$$

MINFREQ() returns the node that is a *CH* and has the least frequency in $MA_{frequency}$.

Pseudocode for the Add Remove repair method (ARrepair) is shown below in Figure 23.

```
ARrepair
Inputs:
c = chromosome
Outputs:
c = chromosome

execute BFS_wsn()//record shortest paths
while(RED()=false)
{
    mostFrequentB = MAXFREQ()
    convert mostFrequentB to cluster head
}
while(BLUE()=false)
{
    mostFrequentB = MAXFREQ()
    convert mostFrequentB to cluster head
}
while(RED()=true AND BLUE()=true)
{
     leastFrequentCH = MINFREQ()
     convert leastFrequentCH to member node
}
convert last converted node to cluster head
return c
```

Figure 23. Algorithm ARrepair

The algorithm was tested using the same 44-node network as above. The results are shown below in Table 6 using the Elite selection method and mutation rate at 5%.

	20%	40%	60%	80%	Optimal
Fitness Value	204	200.8	201.8	203.5	196
Iterations	9.7	11.7	12.4	10.7	n/a

Table 6. Results ARrepair

As shown, the results of this repair method were significantly better than the previous attempts. ARrepair was able to attain similar fitness values while reducing the number of iterations required by an order of magnitude. Although successful in finding a near optimal solution and reducing iterations, this repair process was computationally very costly. This was

more than balanced in running time because the required iterations with ARrepair could be reduced dramatically. Compared to the original algorithm running for 500 iterations versus ARrepair running on the bottom 40% for 20 iterations, ARrepair resulted in similar fitness values but ran in one quarter of the time as shown in Figure 24.

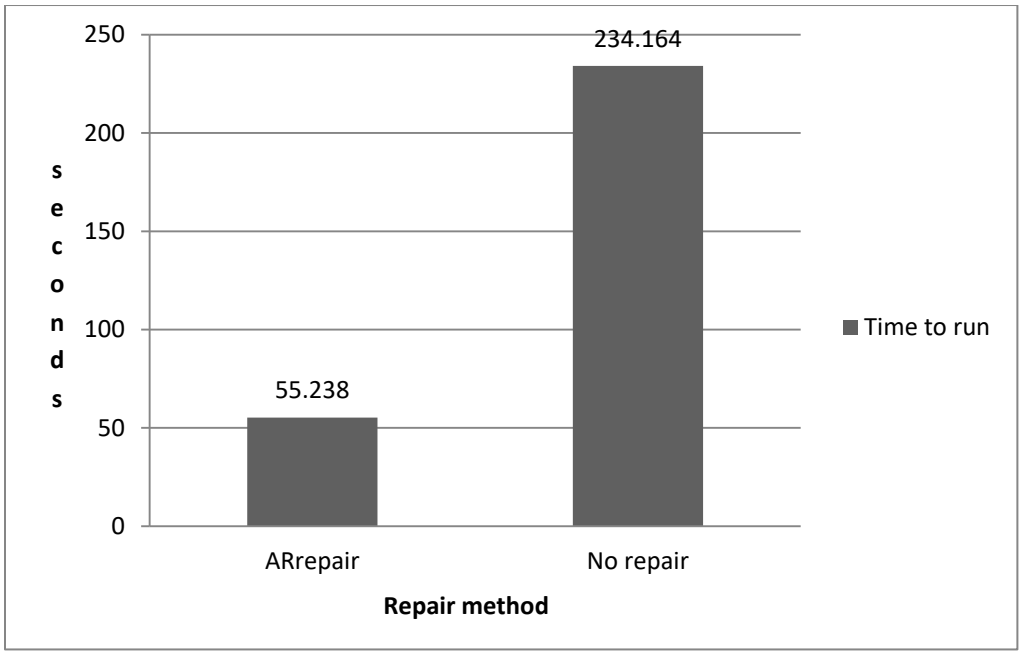

Figure 24. Timing Results

However, as shown in the pseudocode, the RED() and BLUE() subroutines had to be re-run multiple times. Therefore a less complex approach was developed using BFS_{wsn}.

Repair method 4.

In order differentiate the repair methods, the process described above will be referred to as Add-Remove repair (ARrepair) and the process described below will be referred to as Breadth-First Search repair (BFSrepair). Initially BFS_{wsn} was run on the entire network in order to determine the shortest path from each node to the sink. During this process the children of each node were recorded. Recall from the BFS_{wsn} description that as a node n_e was evaluated,

any nodes that were its neighbors and were added to the open set were considered children of n_e. After BFS_{wsn} was run, any node with no children was made a member node and nodes with one or more children were made cluster heads. This approach was successful in early testing on the smaller networks of 6 and 8 nodes. However when tested on the 44-node network, The BFS approach results were significantly worse than ARrepair.

Research into the cause showed that the ordering of the nodes used was creating many more cluster heads than were required for the network to be feasible. Therefore various methods were tested to improve the results. First was to separate the nodes of each level of the search. This approach is shown below in Figure 25.

```
BFSsep(test separating subsequent evaluations)
Inputs:
sink node
N = {set of network nodes to traverse}
Outputs:
closed = {set of nodes traversed by algorithm}


1 open = {empty set}
2 closed = {empty set}
3 sink_level = 1
4 currentlevel = 0
5 add sink to open
6 while(open ≠ Ø)
7 {
8       ne = first node in open
9       if(ne_level ≠ currentlevel)
10      {
11            currentlevel = currentlevel +1
12      }
13      else
14      {
15            ne = node in open greatest sum distance from nodes in closed 16
              where n_level = currentlevel
17      }
18    remove all children from ne
19      for(each n in N)
20      {
21            if(MA_connect(ne,n)AND(n not in { open, closed }))
22            {
23                  n_level = currentlevel +1
24                  add n to children of ne
25                  add n to open
26            }
27      }
28      remove ne from open
29      add ne to closed
30 }
31 return closed
```

Figure 25. Algorithm BFSsep

First all of the child nodes of the sink (level 1) were identified and added to the list of open nodes as shown in lines 19 through 27. These child nodes were all considered level 2 (line 23). Then the sink was removed from the open list and added to the closed list in lines 28 and 29. The first node of level 2 had its children identified and added to the open list as level 3 nodes in the same manner. The remaining level 2 nodes in the open listed were checked for which node was the greatest total distance from all level 2 nodes in the closed list as shown in lines 15 and

16. This farthest node was then processed next, adding its children to the open list as level 3 nodes and then the farthest node was removed from the open list and added to the closed list. This process was repeated for all level two nodes until they were all removed from the open list. Then the overall process was repeated for each subsequent level until all nodes of the network had been searched.

This method was tested on various networks and was found to typically provide equivalent fitness values to the ARrepair method on the 6 and 8-node networks, but was approximately 10% worse on the 44-node network.

In evaluating this approach, it was found that the repair created the same network configuration every time, which logically is what should have happened. In order to remedy this, another version, BFS_{rand}, was implemented where the nodes at each level were randomly shuffled prior to evaluation. This approach is shown below in Figure 26.

```
BFSrand

Inputs:
sink node
N = {set of network nodes to traverse}
Outputs:
closed = {set of nodes traversed by algorithm}

open = {empty set}
closed = {empty set}
sink_level = 1
currentlevel = 0
add sink to open
while(open ≠ Ø)
{
     ne = first node in open
     if(ne_level ≠ currentlevel)
     {
          currentlevel = currentlevel +1
          randomly reorder nodes in open where node_level = currentlevel
     }
     ne = first node in open
     remove all children from ne
     for(each n in N)
     {
          if(MA_connect(ne,n)AND(n not in { open, closed }))
          {
               n_level = currentlevel +1
               add n to children of ne
               add n to open
          }
      }
      remove ne from open
      add ne to closed
}
return closed
```

Figure 26. Algorithm BFSrand

This improved the performance of the BFSrepair significantly. After additional testing it was found that randomly selecting the order of node processing at each level was superior to selecting the farthest node as described above. It was also tested if running the BFSrand at each iteration would improve the results. There was an increase in performance which was expected since it allowed greater variation in the order of node processing and therefore increased the size of the search space. Therefore the final version of BFSrepair is shown below in Figure 27.

```
BFSrepair
Inputs:
c = chromosome
N = {nodes of WSN}
Outputs:
c = chromosome

execute BFS_rand()//record node children
for(each n in N)
{
     if(n has any children)
     {
          set the gene in c that corresponds to n as a cluster head
     }
     else
     {
          set the gene in c that corresponds to n as a member node
     }
}
return c
```

Figure 27. BFSrepair

The results of the final BFSrepair algorithm compared to ARrepair are shown below in Table 7. Both were run using Elite selection and 5% mutation rate. Also both were run using the repair method on the bottom performing 40% of chromosomes at each iteration.

	BFSrepair	**ARrepair**	**Optimal**
Fitness Value	199.2	200.8	196
Iterations	10.8	11.7	n/a

Table 7.Results Repair Comparison

As shown the fitness values and iterations were very similar with the two methods with BFSrepair performing slightly better. Also both repair methods were within 2.4% of the estimated optimal value (estimation method detailed in Appendix B). The final comparison between the methods at this stage was done on the processing time required. All tests were performed on the same pc under the same conditions. The two repair methods were run as they had been previously; 50 chromosomes, Elite selection, 5% mutation, 20 iterations, and repairing the bottom 40% of chromosomes each iteration. The original algorithm with no repair was run

using 100 chromosomes, Elite selection, 5% mutation, and 500 iterations. These conditions were required by the original algorithm in order to reach the same approximate fitness value as the two repair methods.

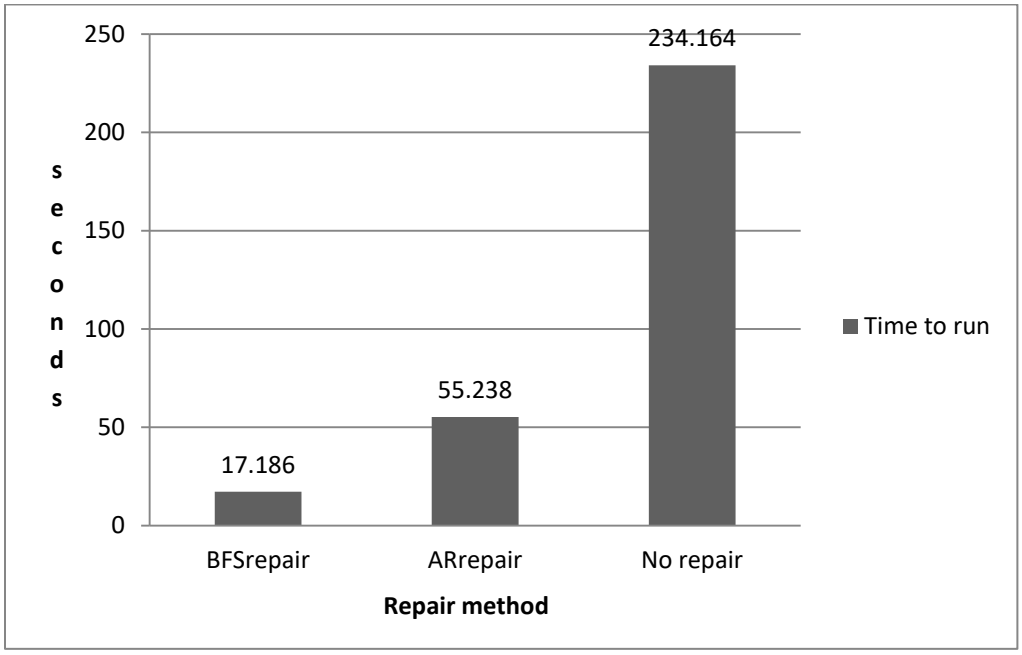

Figure 28. Timing Comparison

As shown both repair methods required significantly less time as compared to the original algorithm with no repair.

The results of Phase 2 were promising for the remaining phases. Two successful repair methods were developed and tested. It was found that the two repair methods provided similar results although ARrepair had higher complexity versus BFSrepair. It was also shown that using repair methods could significantly reduce the number of iterations required to reach a near-optimal solution. Therefore both repair methods would be tested in subsequent phases as the complexity of the problem was increased.

Phase 3. In Phase 3 the values for energy usage were added back in to the target function (equation 5).

$$f(E_{recv}, CH) = \sum_{i=1}^{n}[x_i * k + y_i] * E_{recv} + e|CH| \quad (5)$$

In order to simulate the energy usage of nodes in the network a system of rounds was implemented within the calculation for energy usage. Ten rounds were used for Phase 3 testing. Each node was assigned a binary value for each round indicating whether or not it sent a message that round. Table 8 is an example of the data for the 6-node network where r1 through r10 represent each round and whether or not a message was sent from the corresponding node that round. The energy calculation method is shown below in Figure 29.

ID	x	y	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10
1	0	1	0	0	1	0	1	1	0	1	1	1
2	0	2	1	0	0	0	0	1	0	0	0	1
3	0	3	0	0	1	1	1	0	1	1	0	1
4	0	4	1	1	0	1	0	1	0	1	0	1
5	1	3	0	1	1	0	0	1	1	0	1	1
6	1	1	0	1	0	0	0	1	0	1	1	1
-1	0	4.8										

Table 8. WSN Nodes with Rounds

```
Energy calculation
Inputs:
c = chromosome
N = {nodes of WSN}
CH = {cluster heads}
rounds = number of rounds
e_send = energy to send message
e_recv = energy to receive message
e_main = energy to maintain cluster head
Outputs:
Total energy used by network

set energy used by all nodes to 0
for(each r in rounds)
{
     for(each n in N)
     {
          if(n sending message during r)
          {
               add e_send to n_energyused
               NP = {nodes in path from n to sink}
               for(each np in NP)
               {
                    add e_send to np_energyused
                    add e_recv to np_energyused
               }
          }
     }
     for(each ch in CH)
     {
          add e_main to ch_energyused
     }
}
for(each n in WSN)
{
     totalEnergy = totalEnergy + n_energyUsed
}
return totalEnergy
```

Figure 29. Energy Calculation

The percentage of nodes sending messages each round varies greatly in the literature from Drugan et al. (2011) at 25% to Henzelman et al. (2002) at 100%. This variation is logical since it depends on the purpose of the sensor network. For example, seismic detectors may only send data if an event is detected whereas temperature sensors would send messages every cycle

as is acknowledged in Henzelman et al. (2002). Given these values it was decided to test the algorithm at various percentages to determine if the value affected the performance of the algorithm. The initial values were 25%, 50%, 75%, and 100%.

As noted previously, nodes not sending or receiving could be assumed to enter a sleep mode and use no significant energy (Lochin et al., 2003). This was the case assumed with member nodes. Cluster heads were assumed to consume 1 joule (J) per round for the value of *e* in the energy function. Although this is much higher than real world consumption, it is used here as a relative measure and as long as it is held constant across tests does not affect the outcome. The energy consumed by a node sending a message was shown to be 65% greater than at idle, and receiving a message was 25% higher than at idle (Lochin et al, 2003). Using these values along with 1J for *e* yields a value of 1.25J for E_{recv} and 1.65J for E_{send} and therefore 1.32 for the value of *k*.

During early testing, it was found that energy usage would vary significantly based on the timing of sending messages through the overlay network. In Figure 10 below it could be assumed the overly network functions in two ways. Recall in Figure 10 the squares represent cluster heads and the circles represent member nodes.

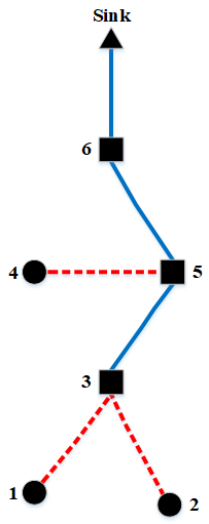

Figure 10. 6-Node WSN

One option assumes aggregation of messages at cluster heads and synchronization of the network. For example, the messages from nodes 1 and 2 are aggregated at node 3 and then forwarded through the overlay network to the sink. The message from node 4 is first sent to node 5 and then also forwarded to the sink. If it could be assumed that the overlay network waits for the messages from nodes 1, 2, and 4 to be sent before forwarding to the sink, then this would reduce the traffic in the overlay network and thereby reduce the energy usage of the network. However this coordination would be difficult to achieve in field conditions and the delay in messages being sent to the sink may unacceptable for networks requiring real-time data such as military applications. Therefore it was assumed going forward that all networks operated in this real-time mode. As messages were sent each round they would not be aggregated at the cluster heads and would be forwarded through the overlay network without waiting for other nodes.

Another finding during early testing was that when energy usage was added to the fitness value, it was not always optimal to connect a member node to the nearest cluster head. For example, again referring to Figure 10, node 4 is connected to the nearest cluster head, node 5,

based on Euclidean distance. However, if node 4 was connected to node 6 it would reduce the total number of hops to the sink for its message. Although at this stage connecting a member node to the cluster head nearest a sink was preferable, it was unknown if this would always be preferable, considering balancing energy across nodes. Therefore the connection method was chosen at random for each member node. The options were, from the cluster heads within transmission range, connecting to the nearest cluster head or connect to the cluster head nearest to a sink. Although the Elite selection method was already retaining the top 20% of the nodes, the top performing node also had its fitness value and connection strategy passed on to the next generation. This was because two chromosomes could identify the same cluster heads, but the connection strategy could make one superior to the other.

The first testing was done on the same 6-node network from Phases 1 and 2. Since it was shown that mutation rate and selection method had little to no effect on the algorithm using repair methods, these were held constant using 5% for the mutation rate and Elite selection for all testing in Phase 3. The 6-node network was tested with 20 chromosomes.

	25%	**50%**	**75%**	**100%**
BFSrepair – Fitness value	112	204	254	392
ARrepair – Fitness value	112	204	254	392
BFSrepair – Iterations	1.9	1.8	1.7	1.8
ARrepair – Iterations	1.8	1.9	1.8	1.8

Table 9. Varying Message Percentage

As shown above in Table 9, both the fitness scores and the iterations were nearly identical. This was to be expected since both repair methods were able to converge to optimal solutions for the simple 6-node network. Next the repair methods were tested on the 44-node network from Phase 2. These tests were again run at 5% mutation and using Elite selection, but with 50 chromosomes for the larger network.

	25%	**50%**	**75%**	**100%**
BFSrepair – Fitness value	784	1630	2035	2900
ARrepair – Fitness value	828	1718	2240	2978
BFSrepair – Iterations	12.8	10.8	12.5	10.4
ARrepair – Iterations	12.6	12	13.7	11.4

Table 10. Varying Message Percentage 44-Node Network

As shown the BFSrepair fitness values were 4% to 8% better than the ARrepair values depending on the percentage of nodes sending messages each round. The iterations required by the two methods were again practically identical. The superior performance of BFSrepair was likely due to the even, radial configuration of the network nodes. This would be explored further in Phase 4 with larger networks using randomly-positioned nodes.

As shown above, Phase 3 showed successful implementation of the two repair methods with energy values added back in to the target function. The method for deciding which cluster a node was a member of was modified to increase the search space. The BFSrepair method was shown to be slightly superior. Both methods would continue to be tested in subsequent phases.

Phase 4. This phase included the addition of gateway nodes to the network configuration and testing the algorithm on larger, randomly-generated networks. Recall Figure 10 below.

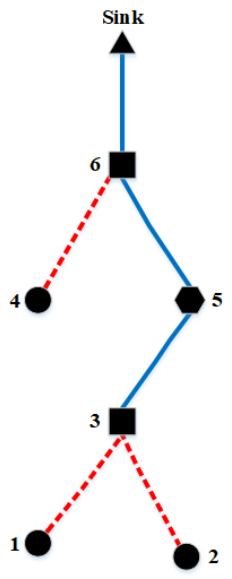

Figure 10. WSN with gateway node

The gateway node in the figure above is depicted as a hexagon (node 5). The addition of the gateway node reduces the required number of cluster heads without reducing the connectivity of the network. Gateway nodes were considered part of the *BLUE* graph for network feasibility, but because gateway nodes do not manage any member nodes, they do not incur the additional energy cost associated with cluster heads.

The encoding of the gateway nodes within each chromosome was done within the existing framework by making each gene's value either 0, indicating member node, 1, indicating cluster head, or 2, indicating gateway node. Early testing showed that adding gateway nodes when initially creating the chromosomes did not improve the performance of the algorithm. It was then determined that adding gateway nodes during the repair processes was preferable. Both repair methods, ARrepair and BFSrepair required modification to implement gateway nodes and there was also minor modification of the fitness calculation and the mutation process.

The ARrepair method that implements gateway nodes is shown below in Figure 30.

```
ARrepair
Inputs:
c = chromosome
CH = {cluster heads}

Outputs:
c = chromosome

1 execute BFSwsn()//record shortest paths
2 while(RED()=false)
3 {
4      mostFrequentB = member node most frequent in shortest paths
5      convert mostFrequentB to cluster head
6 }
7 while(BLUE()=false)
8 {
9      mostFrequentB = member node most frequent in shortest paths
10     convert mostFrequentB to cluster head
11 }
12 while(RED()=true AND BLUE()=true)
13 {
14      leastFrequentCH = cluster head least frequent in shortest paths
15      convert leastFrequentCH to member node
16 }
17 convert last converted node to cluster head
18 for(each ch in CH)
19 {
20      if(ch has no member nodes)
21      {
22            convert ch to a gateway node
23      }
24 }
25 return c
```

Figure 30. Algorithm ARrepair with Gateway Nodes

As shown in lines 18 through 24, the change to the repair method involved checking each cluster head for member nodes. If a cluster head had no member nodes to manage, it was converted to a gateway node. The modification to the BFSrepair method is shown below in Figure 31.

```
BFSrepair
Inputs:
c = chromosome
N = {nodes of WSN}
Outputs:
c = chromosome

1 execute BFS_rand()//record node children
2 for(each n in N)
3 {
4       if(n has any children)
5       {
6             n_children = {children of n}
7             if(all nodes in n_children have children)
8             {
9                   set the gene in c that corresponds to n as a gateway node
10            }
11            else
12            {
13                  set the gene in c that corresponds to n as a cluster head
14            }
15      }
16      else
17      {
18            set the gene in c that corresponds to n as a member node
19      }
20 }
21 return c
```

Figure 31. Algorithm BFSrepair with Gateway Nodes

As shown, the BFSrepair method now checks the number of children for each node. Nodes with children have the potential to be gateway nodes as long as all of the child nodes are cluster heads or gateway nodes. This is determined by checking if the child node has any children as shown in line 7. Nodes with no children must be member nodes.

The fitness calculation was modified to consider gateway nodes as part of the *BLUE* graph for the feasibility check. Mutation was modified because with the addition of gateway nodes there were now three potential states for each node; member node, gateway node, and cluster head. Therefore mutation was configured so that there was an equal random chance that a gene would mutate to either of the other possible states. For example a node that was a cluster

head had an equal chance of mutating to either a gateway node or a member node, whereas a member node could mutate to either a gateway node or a cluster head.

Testing the algorithms with the gateway node modifications was done initially on the same 44-node network that was used in Phase 3. It was shown in Phase 3 that the percentage of nodes sending messages each round did not affect the relative performance of the repair algorithms, therefore in this phase the algorithms were only tested on the network with 50% of the nodes sending messages each round. Also only Elite selection was used and mutation was fixed at 5%. In order to increase the search space 100 chromosomes were used, running for 60 iterations. The results are shown below in Table 11.

	Fitness	**Iterations**	**#CHs**	**#GNs**
BFSrepair – No Gateway	1621.0	13.5	13.1	0
ARrepair – No Gateway	1638.4	22.9	12.5	0
BFSrepair – with Gateway	1613.7	11.5	12.2	1.9
ARrepair – with Gateway	1609.7	17.7	10.0	6.8

Table 11. Results, Gateway Nodes

As shown, the use of gateway nodes was able to slightly reduce the energy usage of the network and the ARrepair method had a greater reduction. This was due to the BFSrepair algorithm using slightly more cluster heads and therefore creating shorter paths in general to the sink. In ARrepair there greater overlay path inefficiencies and therefore there were more cluster heads that could be converted to gateway nodes resulting in greater energy savings. However overall it was seen that even with the greater improvement from gateway nodes in ARrepair, BFSrepair still had preferable fitness values. This was because the total energy contribution of

cluster head maintenance was relatively low. Using the maximum number of cluster heads from the average of 10 runs, 13.1 as shown above, in this network run the cluster heads only contributed 8% of the total energy use. Therefore the superior arrangement for message sending had a much greater impact on total energy use than the number of cluster heads.

It could also be seen in Table 11 that the use of gateway nodes slightly decreased the number of iterations required to reach the optimal solution. This was unexpected since gateway nodes increase the search space and intuitively should require more iterations. This result was likely due to the very regular node arrangement of this test network.

The next set of tests involved creating two random 100-node networks. The method for creating the random networks is shown below in Figure 32.

```
Network creator
Inputs:
numNodes = number of nodes
gridSize = the square dimension of the network area
tr = transmission range of the nodes
minDistance - the minimum Euclidean distance between nodes
Outputs:
N = set of nodes for WSN

xCoord = random between zero and gridSize
yCoord = random between zero and gridSize
create sink with xCoord and yCoord
add sink to N
while(number of nodes in N < numNodes)
{
      xCoord = random between zero and gridSize
      yCoord = random between zero and gridSize
      newNode = create node with xCoord and yCoord
      if(minDistance < newNode < tr relative to any node in N)
      {
            Add newNode to N
      }
}
return N
```

Figure 32. Network Creator

As shown in Figure 32, while the nodes were randomly located, they had to be within transmission range of another node and also no closer than a predetermined distance to another node (line 10). This is reasonable because the ideal network would not have nodes too close together otherwise they would produce duplicative data and nodes beyond transmission range of the network would be of no use. In the first network the transmission range was set at 1.5 units as it had been in previous networks. The grid size was set to 100 and the minimum distance was set to 0.25 providing a relatively dense network of 100 nodes. The nodes are shown below in Figure 33.

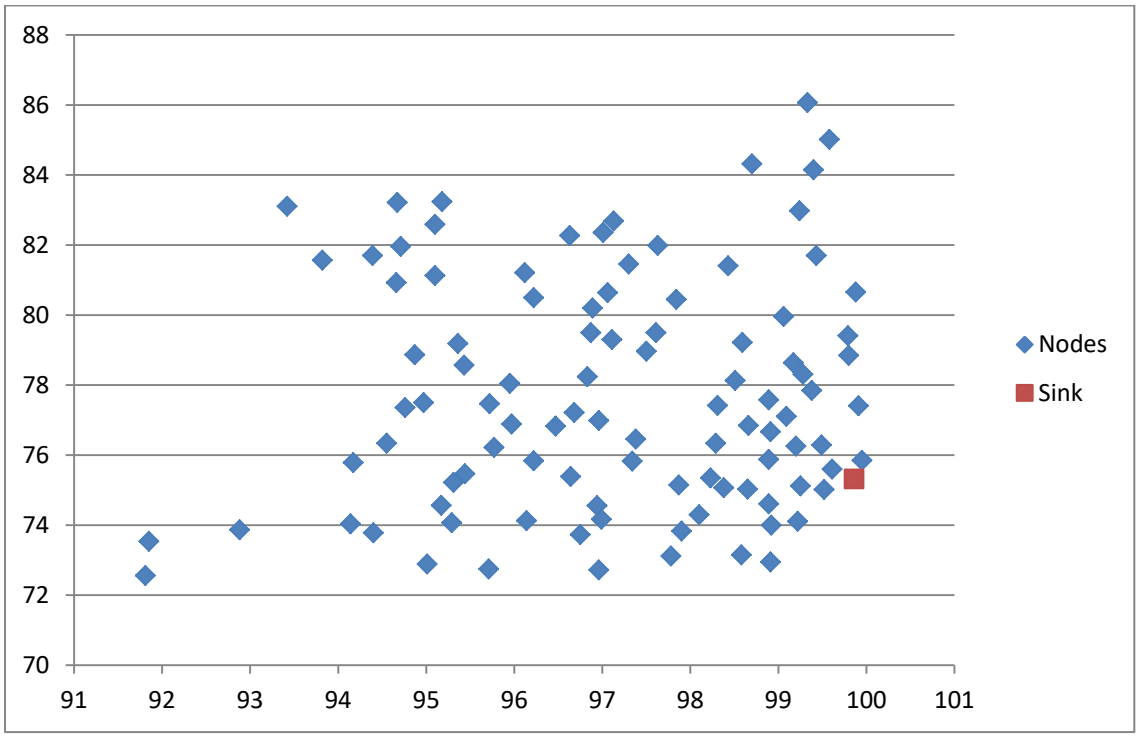

Figure 33. 100-Node Network

The results of testing the two repair methods on this network are shown below in Table 12. The network was again tested with 50% of the nodes sending messages each round. Also only Elite selection was used and mutation was fixed at 5%. For this larger network 200 chromosomes were used running for 60 iterations.

	Fitness	Iterations	#CHs	#GNs
BFSrepair – No Gateway	6852.3	41.2	38.6	0
ARrepair – No Gateway	6966.5	53.4	38.4	0
BFSrepair – with Gateway	6668.8	50.0	24.2	17.7
ARrepair – with Gateway	6718.3	44.4	14.2	47.7
Optimal (estimated)	6667			

Table 12. Results, 100-Node Network

As shown, the BFSrepair method continued to provide better results than the ARrepair method with the 100-node random network. The estimated optimal value for this network was also included for comparison. The estimation method is included in Appendix B. As shown the ARrepair method was within 4.5% of the optimal and the BFSrepair was within 2.8% of the optimal. Therefore it could be seen that both repair methods provided good solutions based on fitness value.

It was also shown that the use of gateway nodes had a more significant impact on the larger network, reducing energy usage on average by 3% versus 1% on the 44-node network. While this improvement may not seem significant, in this network cluster heads only contributed on average 5.6% of the energy usage, therefore this improvement does represent a significant improvement in the amount of energy used by cluster heads. It was also shown in this test that the use of gateway nodes increased the required number of iterations as was expected.

For the next test, another random 100-node network was created, but this time with the minimum distance between nodes set to 1.00. This created a sparser network as shown in Figure 34.

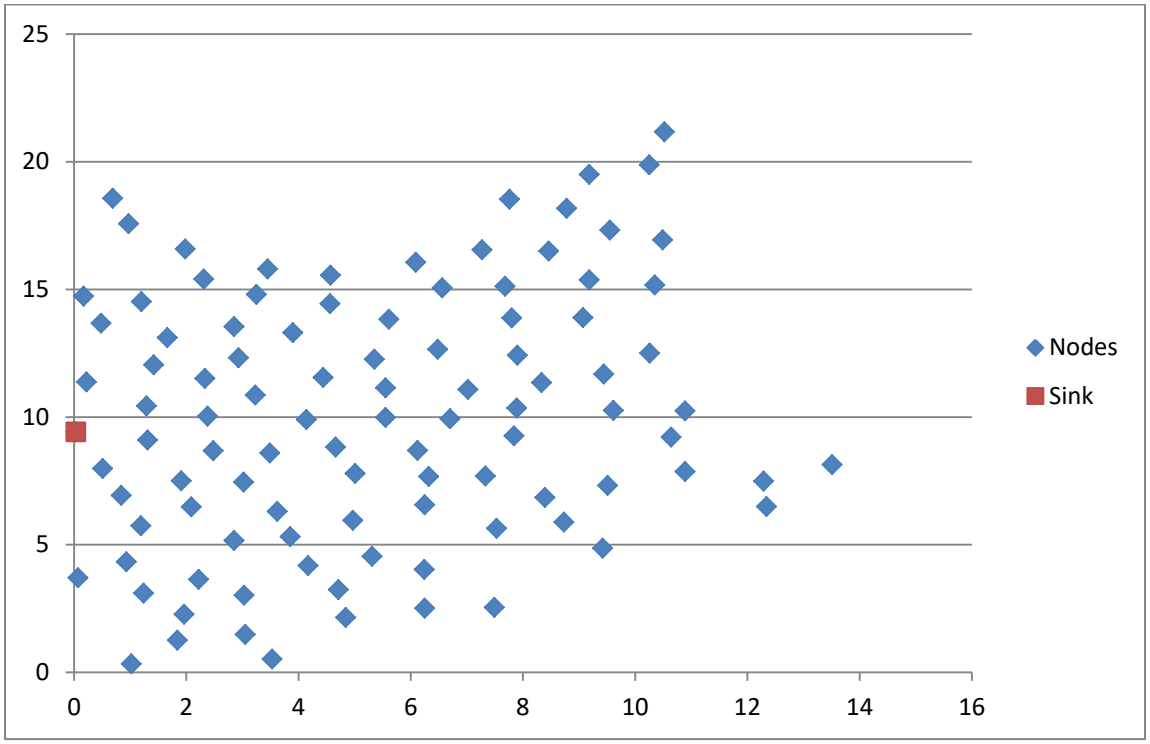

Figure 34. Sparse 100-Node Network

The results with this network were similar to those with the denser network shown in Table 12. The overall energy usage of the network was increased, which was expected given that a sparser network would require more nodes in the overlay network (*BLUE* graph). The results are shown below in Table 13.

	Fitness	Iterations	#CHs	#GNs
BFSrepair – No Gateway	10602.6	35	63.1	0
ARrepair – No Gateway	10600.5	39.0	63.7	0
BFSrepair – with Gateway	10218.0	13.1	25.0	45.8
ARrepair – with Gateway	10174.2	17.2	17.8	62.1
Optimal (estimated)	10490			

Table 13. Results, Sparse 100-Node Network

As shown it was seen that the BFSrepair and ARrepair methods provided nearly equal results in the sparse network. The use of gateway nodes also had a slightly more significant impact on the sparser network, on average improving the fitness value by 3.8%. Again while a relatively small reduction in the total network energy usage; this was a significant reduction in the maintenance energy of the network. In the BFSrepair algorithm the maintenance energy was reduced by 60% and in the ARrepair algorithm it was reduced by 72%. The only maintenance being performed by the cluster heads in this work is listening for and routing messages. In networks where cluster heads perform additional work such as message aggregation and intrusion detection this energy savings would be more significant versus the overall network energy.

The estimated optimal value for the sparse network was also included in Table 13 for comparison. The estimation method is included in Appendix B. As shown the ARrepair method was within 1% of the optimal and the BFSrepair was within 1.1% of the optimal. Therefore it could be seen that both repair methods provided good solutions based on fitness value.

Overall in Phase 4, it was shown that both repair algorithms demonstrated a slight overall improvement in fitness scores using gateway nodes. It was shown that while the overall

improvement was typically 1%-4% this represented a large percentage of the energy used by cluster head maintenance. It was also shown that both repair methods were effective on larger, random networks by comparison to the estimated optimal values. Finally the BFSrepair method provided slightly better results versus the ARrepair method with the exception of the 100-node sparse network where the two methods were practically equal. Therefore research continued with work on both repair methods in Phase 5.

Phase 5. The primary work of this phase was to implement energy balancing among the nodes of the network. The intent of this was to improve the lifetime of the network by reducing the energy usage what were termed popular nodes. These were nodes that were involved in a majority of the network communications. For example recall figure 10 below:

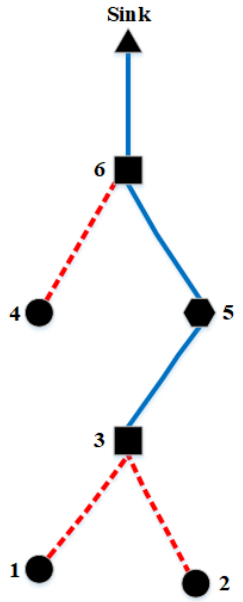

Figure 10. WSN with gateway node

All messages sent by any node in the network had to be received and then sent by node 6. Therefore node 6 was deemed a popular node. In the network shown in Figure 10 above, node 6

will always be the first node to run out of energy due to all of the network traffic running through it. In a small network as shown there is no method to improve this situation. However in larger networks where there are many possible paths to the sink, traffic can be routed to different nodes, thereby reducing the energy usage of the individual nodes. Therefore the BFSrepair algorithm was modified to evaluate the energy usage in the nodes at the end of each round. The pseudocode for the modification is shown below in Figure 35.

```
BFSrepair with energy balance

Inputs:
sink node
N = {set of network nodes}
Outputs:
N = {set of network nodes}

open = {empty set}
closed = {empty set}
holding = {empty set}
sink_level = 1
currentlevel = 0
add sink to open
while(open ≠ Ø)
{
     ne = first node in open
     if(ne_level ≠ currentlevel)
     {
          currentlevel = currentlevel +1
     }
     for(each n in N)
     {
          if(MA_connect(ne,n)AND(n not in { open, closed }))
          {
                n_level = currentlevel +1
                add n to open
          }
      }
      remove ne from open
      add ne to closed
}
foreach(node n in wsn)
{
      Identify possible parents where(level==i-1)AND
      (MA_connect=true)AND(parent!=n)
      Find parent p with highest energy
      Add n to children of p
}
return closed
```

Figure 35. Algorithm BFSrepair with Energy Balance

As shown, the initial part of BFSrepair is the same as it was previously. However as shown in lines 25 through 31, after levels are assigned to all of the nodes, children are assigned based on the remaining energy in the nodes. This proved to be a successful strategy for balancing the energy usage of the nodes.

The modification of the ARrepair algorithm was straightforward. Instead of adding and removing cluster heads based on frequency, it was modified to select nodes with the least energy usage to be preferential to be converted to cluster heads. Conversely in the section of the algorithm where it converted cluster heads to member nodes, those with the greatest energy usage were preferred for conversion.

The modified algorithms were first tested on the 44-node network shown in Figure 17 The tests were run using 80 chromosomes and as was done previously, 10 runs were completed and the average recorded. The results are presented in Figure 36 below using the repair algorithms with and without the energy balance component.

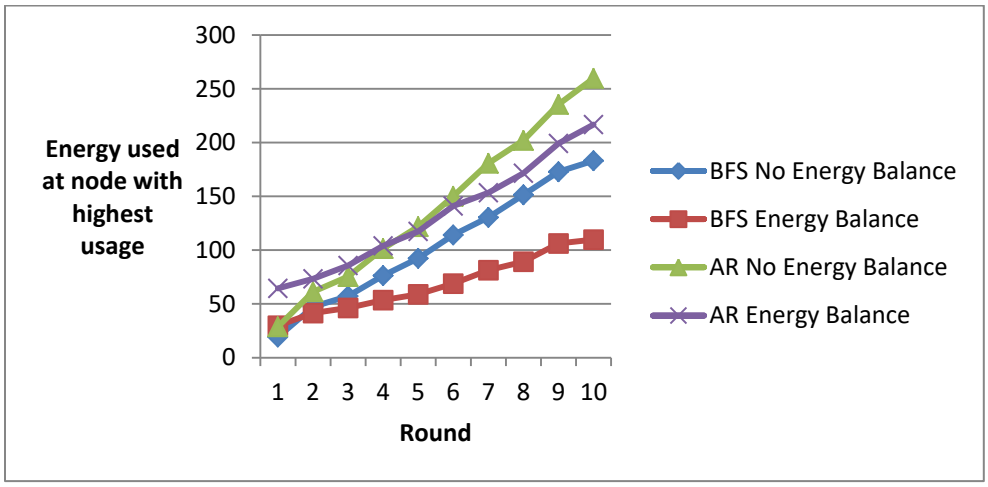

Figure 36. Results, 44-Node Network

As shown, the energy balance modification was able to reduce the energy used at the most popular nodes of the network. The difference at the 10th round was such that the BFSrepair energy balance algorithm was able to reduce the energy used by the most popular node by 40%. While it showed and improvement of 16%, the ARrepair algorithm performed poorly in this test as compared to the BFSrepair algorithm.

Next the algorithms were tested on the 100-node network shown in Figure 33. For this test 160 chromosomes were used and the results showed a greater improvement than with the 44-node network.

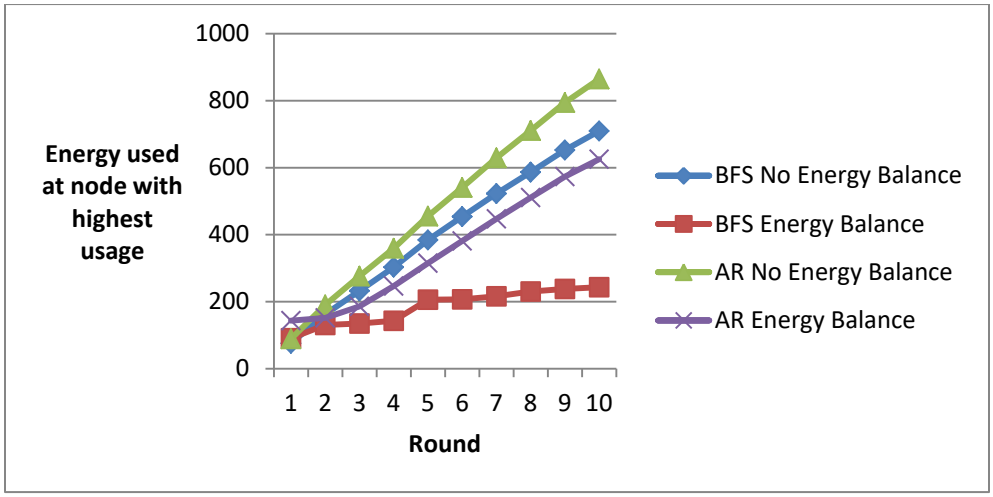

Figure 37. Results, 100-Node Network

As shown, the BFSrepair algorithm using energy balance had used 66% less energy at the most popular node at round 10. The ARrepair algorithm showed a 21% improvement. Again the ARrepair algorithm performed poorly as compared to the BFSrepair algorithm. This was expected because the ARrepair algorithm was initially based on frequency of node usage and therefore tended to drive more traffic to fewer nodes. Changing ARrepair to be based on energy usage improved the results but still did not reduce energy usage at individual nodes as well as BFSrepair. Attempts were made to improve the performance of ARrepair, including using a

larger number of chromosomes and ranking nodes for conversion to or from cluster head based on a combination of energy usage and frequency in shortest path trees. None resulted in a significant improvement. The ARrepair algorithm could be further modified to purposefully change nodes to cluster heads randomly or based on another criteria to distribute the load more evenly, but such changes would move the ARrepair algorithm to be more like the BFSrepair algorithm and therefore were not deemed worthwhile.

This phase showed the while both repair algorithms could be modified to take into account energy usage at individual nodes, the BFSrepair algorithm was superior in this regard. This, combined with earlier results, showed that the BFSrepair algorithm was the superior algorithm and should be the one extended in future work.

Conclusions, Implications, Recommendations, and Summary

Conclusions

It has been shown that clustering the nodes of a WSN can reduce the energy usage of the network (Cheng, 2012; Abbasi & Younis, 2007). The clustering involved selecting nodes to act as cluster heads and selecting other nodes as member nodes to be managed by each cluster head. The purpose of this work was to accomplish the clustering of the nodes using machine learning, specifically a genetic algorithm.

The early phases of this work were able to accomplish the goal of clustering the nodes and provide an optimal selection of cluster heads and associated member nodes known as the network configuration. This was accomplished using simplified networks and the results were verified to be the optimal configuration possible. Therefore the feasibility of using a genetic algorithm to cluster the nodes of the network was verified.

Additional features of the genetic algorithm were tested to refine the clustering algorithm. Different selection methods including elite, roulette-wheel, linear rank, and tournament were all tested. It was found that generally the selection method did not significantly affect the outcome other than lower performance achieved using roulette-wheel as evidenced by less optimal fitness values. Various mutation rates were tested and also found to have minimal impact on the fitness value achieved by the network.

Chromosome repair was investigated with several methods attempted and two were found to be successful. The first method, ARrepair, functioned through adding cluster heads based on frequency of use until the network was feasible and then the least frequently used cluster heads were removed. The second successful method, BFSrepair used a modification of

the breadth-first-search algorithm to create a tree-like structure of the nodes and then select cluster heads and their members based on which nodes were children of others. It was found that the two repair methods provided similar results, with BFSrepair typically better based on fitness value with variation depending on the number and density of nodes within the network. It was also found that using repair methods could significantly reduce the number of iterations required to reach a near-optimal solution. It was found that although the repair algorithms were more computationally complex than the genetic algorithm alone, a near-optimal solution could be found faster due to the reduced number of iterations and number of chromosomes required using the repair methods.

A complex target function used to calculate the fitness value of each chromosome was implemented. The target function took into account the energy used at each node in the network to send and receive messages as well as the additional energy required for maintenance of the cluster at the cluster head. It was found that both repair methods were able to produce near-optimal results using the complex, energy-based target function to calculate the fitness value of each chromosome.

The genetic algorithm and associated repair methods were tested on increasingly complex networks using up to 100 nodes in randomly generated locations. The networks tested also varied in the density of the nodes. It was found that the results from less complex networks were repeated for more complex networks. Also as the number of nodes increased, the difference in fitness value achieved by the different repair methods also increased to as much as 10% with BFSrepair found to be typically superior. However the two methods were found to be equal on a 100-node sparse network.

It was also determined that genetic algorithm and repair methods were able to implement the concept of gateway nodes. Gateway nodes acted as cluster heads for the purposes of network communication, but did not manage any member nodes and therefore did not incur the extra energy costs of cluster heads. It was found that gateway nodes could be implemented within the framework of both repair methods and that using them improved the fitness values by 2% to 4% depending on the network.

Finally it was shown that both repair algorithms could be modified to balance the energy usage of individual nodes, thereby prolonging the lifetime of the network. In this regard the BFSrepair algorithm was found to be superior reducing the energy usage as much as 66% compared to BFSrepair without energy balance. BFSrepair with energy balance also resulted 64% lower energy usage at the most popular node as compared to ARrepair with energy balance.

In summary it was found that a genetic algorithm could be successfully used to cluster the nodes of a wireless sensor network and thereby improve the energy efficiency of the network. It was also found that chromosome repair methods cold improve the performance of the genetic algorithm and that the genetic algorithm with repair methods was suitable for use across a range of networks.

Implications

This work evaluated the use of a genetic algorithm for clustering the nodes of a wireless sensor network. It also evaluated the use of chromosome repair to improve the performance of the genetic algorithm. Both the genetic algorithm and chromosome repair methods were found to be successful across a range of networks. The methods developed in this work could be the

basis for developing software to manage wireless sensor networks so that the lifetime of the devices and the network as a whole could be prolonged.

There were several constraints put on the problem within the scope of this work: network nodes were assumed to not be mobile, the nodes were assumed to all of a similar type and therefore be identical in their characteristics such as transmission range and energy usage, and the radius of the clusters was constrained to be within the transmission range of the nodes. The work presented would need to be extended to address networks where these constraints were not applicable.

The work presented also did not include networks where there were multiple sinks. However the algorithm presented could be extended to handle such networks. The algorithm could be modified for multiple sinks through first modifying the *BLUE* graph feasibility check. The *BLUE* graph check would have to check connectivity to any sink for feasibility. The calculations of energy usage would have to be modified through having cluster heads communicate with the nearest sink. This would work for a single connected component or multiple sink components or even a combination of both within the same network.

It was shown previously that the p-median problem could be constrained in such a manner as to resemble the problem presented here in clustering wireless sensor nodes. Therefore the work presented here could be extended to include location problems that can be constrained in a similar manner.

Recommendations

As stated above, this work was implemented on a constrained wireless sensor network. These constraints were necessary to reduce the work to a manageable scope. Additional work

should be performed implementing the algorithms on networks without the constraints noted above.

As noted above, the work presented used networks with a single sink. Along with extending this work to networks with multiple sinks, work should be done selecting the placement of the multiple sinks for maximum energy efficiency. This would also allow the work to more closely align with location problems. Work should also be done to expand the presented algorithms so that they work with nodes with varying transmission ranges. Nodes could be configured so that they expend more or less energy to increase or decrease their transmission range respectively. Implementing the algorithms presented to work with these nodes would further increase the energy efficiency of the network.

Also the algorithms developed here should be modified for use with networks where the nodes are mobile. Networks with mobile nodes known as mobile ad-hoc networks (MANETs) represent another area of research. Extending the work presented here to work with MANETs would be a significant challenge but a worthwhile extension in order to improve the lifetime of MANETs given the many applications of MANETs.

Summary

Wireless sensor networks consist of small battery powered nodes that send messages to a sink or base station. The nodes may form an ad hoc network to facilitate communication. They have been shown to have many uses from military to first responders (Abbasi & Younis, 2007). Since the devices used as nodes of the network have limited battery power there have been many proposed approaches to improve the efficiency of resource usage by the nodes. One of the proposed methods is clustering the nodes (Cheng, 2012).

The problem of clustering the nodes in an optimal configuration can be seen as a constrained version of the p-median problem. The p-median problem belongs to a class of problems known as location problems (Reese, 2006). Location problems typically involve the placement of new facilities. The problem is to place the facilities in locations such that the cost of access or distance to the facilities by other members of the set of objects is minimized (Mladenović et. Al., 2007). This has been shown to be an NP-hard problem even in simple configurations (Kariv & Hakimi, 1979).

The wireless sensor network in this problem was constrained to reduce the scope of the problem. The nodes were assumed to be stationary, all of the nodes were assumed to have the same transmission range and use the same energy when sending and receiving. The radius of the clusters was constrained to the transmission range of the nodes.

The nodes of the wireless sensor network were considered vertices on an undirected graph. The algorithm developed in this work created a graph partition, dividing the wireless sensor network into two subgraphs, one that formed the primary communications path for the network (*BLUE*) and a second subgraph where the member nodes were connected to the nodes of the first set (*RED*). In order to be considered feasible these partitions had to provide a communication path for all nodes of the network to a sink node.

Genetic algorithms have been shown in the literature as providing a means to solve the p-median problem (Correa et al., 2004) and for clustering network nodes (Peiravi et al., 2013). Therefore a genetic algorithm was used as the basis for this work. The chromosomes were encoded so that each node in the network corresponded to a gene with a value of 1 indicating the node was cluster head and a value of 0 indicating the node was a member node. The fitness

function for the genetic algorithm was based on the amount of energy consumed by the network. Because the goal of this work was to reduce the energy consumed by the network a lower fitness value was considered preferable. Network configurations that were not feasible were given a maximum value to indicate the poorest performance possible.

The work was divided into phases with each phase building on the work of the previous one. The first phase used small networks and a simplified version of the fitness function. This was done in order to be able to verify the results and confirm that the genetic algorithm was functioning as expected and was a viable method for clustering the nodes of the network. The algorithm was found to be successful on the simple networks, exactly matching the optimal configurations of the networks.

Next the complexity of the network and number of nodes were increased as well as the complexity of the algorithm. Crossover and mutation were implemented within the genetic algorithm along with various selection methods including elite, roulette-wheel, linear rank, and tournament. It was found that neither the selection method nor the mutation rate had a significant effect on the performance of the algorithm in this problem space. It was also found that the algorithm was able to perform well on a larger network of 44 nodes.

Chromosome repair methods were also developed and tested at this stage. Two methods tested were found to be successful. One repaired the chromosome by first measuring the frequency with which each node was used during communication to the sink. The *RED* graph was repaired by first adding cluster heads one at a time using the most frequently used member node and then retesting the *RED* graph until it was feasible. Then the *BLUE* graph was repaired using the same approach. Then the network was further improved by removing unnecessary

cluster heads. The least frequently used cluster head was converted to a member node and then the *RED* and *BLUE* graphs were rechecked. If the network was still feasible then another cluster head was converted. This process was repeated until the network failed and the last converted node was returned to a cluster head. This process was named ARrepair.

The other successful repair method first ran a breadth-first-search on the network. During this process nodes that were discovered when evaluating a parent node were assigned as children of the parent node. Next any node with children was converted to a cluster head and nodes with no children were converted to member nodes. This process was called BFSrepair. Both repair methods were tested on multiple networks of up to 100 nodes and were found to successfully determine near-optimal configurations for the networks.

When tested against the genetic algorithm with no repair method both repair methods were able reach solutions much faster despite being more complex. This was because the repair methods required fewer chromosomes and fewer iterations to reach the solution. The BFSrepair method typically provided solutions that were 4%-10% better than the ARrepair method based on fitness value. The BFSrepair method also ran in less time than the ARrepair method.

Finally it was shown that gateway nodes could be implemented within the repair methods and these successfully reduced the energy consumption of the network 2%-3% depending on the network. Therefore it was found that the genetic algorithm with repair methods was a viable technique for clustering the nodes of a wireless sensor network.

There are many possible future extensions of this work. These include removing the constraints mentioned above and modifying the algorithm to function with the new parameters.

This work could also be extended for use with location problems that could constrained in a similar manner. A more detailed discussion of future extensions follows.

One of the assumptions made in the problem statement in order to simplify the problem in this research was that all nodes would be of a similar type and manufacture. This allowed the assumption that the values for E_{recv}, E_{send}, and k would be the same for all of the nodes. These values could be allowed to vary between nodes within a reasonable range determined from the literature. This would require changes in the way the fitness function calculates the score for each chromosome. The variations would be included in the energy used at each node for sending and receiving messages. If successful the algorithm would be able to make greater use of the nodes with lower E_{recv} and E_{send} in order to prolong the lifetime of the network.

Another extension would be to remove the assumption of a fixed transmission range that is the same for all nodes in the network. This would add two different types of complexity to the problem. The first is that of nodes having fixed but different transmission ranges. This would affect the configuration of the overlay network as well as cluster membership.

Another improvement involving transmission range would be variable transmission range. This would represent a significant change in the way the clustering algorithm would operate. The nodes could be configured so that E_{send} would vary with the transmission range; higher energy would be required for longer distance transmission. It is expected that providing nodes with variable transmission range should allow for a more efficient network versus nodes with fixed transmission ranges. Therefore comparisons could be made to the results in this research.

Another extension would be to introduce multiple sinks to the problem. It was shown in the literature review that there may be situations where multiple sinks are desirable within a WSN and the placement of these sinks can be used to decrease the overall energy usage of a WSN. This would add significant complexity to the problem in two ways. There would no longer be a single fixed sink so the proposed algorithm will need to determine the optimal number of sinks based on energy saving versus the cost of additional sinks within a range of available sinks. The maximum number of sinks would have to be limited, otherwise the optimal solution would always be every node should be a sink. The other complexity is locating the sinks so as to reduce the energy used for transmission and reception of data among nodes and cluster heads. Successful testing of this phase would involve comparison against single sink versions of the algorithm. Intuitively multiple sinks should significantly extend the network lifetime as compared to single sink WSNs. Also the algorithm should be able to generate data used to show limiting returns on energy reduction as sinks are added so that an ideal number of sinks could be determined.

The work presented here assumed that the nodes of the WSN were stationary. Another possible improvement would be to allow node movement such as in mobile ad-hoc networks (MANETS). With mobile nodes the clusters would have to be rebalanced more frequently in order to retain complete communication within the network. Rules would have to be established for when to rebalance the clusters and the energy costs for rebalancing would be included in the target function and therefore also in the fitness function.

A genetic algorithm was the focus of this research since they had been shown to be successful in similar problems in the literature. Another possible extension would be to attempt a solution using another machine learning technique such as convoluted neural network (ConvNet)

As noted in the literature review ConvNets are typically used for image recognition. The input to the ConvNet is typically a grid of values representing each pixel in an image (LeCun, Bengio & Hinton, 2015). The input for the ConvNet in the WSN problem could be a grid of binary variables instead of pixels which should reduce the complexity. For the WSN input a '1' would represent a node while a '0' represents empty space. The research presented in this dissertation could be used to create training and testing sets for the ConvNet.

As shown, there are many potential avenues of expansion of this research. The intent of this work is that it provides a good foundation for these future directions.

References

Abbasi, A. A., & Younis, M. (2007). A survey on clustering algorithms for wireless sensor networks. *Computer communications*, *30*(14), 2826-2841.

Ahmed, Z. H. (2010). Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. *International Journal of Biometrics & Bioinformatics (IJBB)*, *3*(6), 96.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer networks*, *38*(4), 393-422.

Bayraklı, S., & Erdogan, S. Z. (2012). Genetic algorithm based energy efficient clusters (gabeec) in wireless sensor networks. *Procedia Computer Science*, *10*, 247-254.

Blum, J. J., Eskandarian, A., & Hoffman, L. (2004). Challenges of intervehicle ad hoc networks. *Intelligent Transportation Systems, IEEE Transactions on*,*5*(4), 347-351.

Chauhan, N., Awasthi, L. K., Chand, N., & Chugh, A. (2011). A distributed weighted cluster based routing protocol for MANETs. In *Computer Networks and Information Technologies* (pp. 147-151). Springer Berlin Heidelberg.

Cheng, H. (2012, May). Genetic algorithms with hyper-mutation for dynamic load balanced clustering problem in mobile ad hoc networks. In *Natural Computation (ICNC), 2012 Eighth International Conference on* (pp. 1171-1176). IEEE.

Chinara, S., & Rath, S. K. (2009). A survey on one-hop clustering algorithms in mobile ad hoc networks. *Journal of Network and Systems Management*, *17*(1-2), 183-207.

Correa, E. S., Steiner, M. T. A., Freitas, A. A., & Carnieri, C. (2004). A genetic algorithm for solving a capacitated p-median problem. *Numerical Algorithms*, *35*(2-4), 373-388.

Domínguez, E., & Muñoz, J. (2008). A neural model for the p-median problem. *Computers & Operations Research*, *35*(2), 404-416.

Drugan, O. V., Plagemann, T., & Munthe-Kaas, E. (2011). Detecting communities in sparse MANETs. *IEEE/ACM Transactions on Networking (TON)*, *19*(5), 1434-1447.Er, I. I., & Seah, W. K. (2010, October). Adaptive cluster-based approach for reducing routing overheads in MANETs. In *Communications (APCC), 2010 16th Asia-Pacific Conference on* (pp. 279-284). IEEE.

Er, I. I., & Seah, W. K. (2010, October). Adaptive cluster-based approach for reducing routing overheads in MANETs. In *2010 16th Asia-Pacific Conference on Communications (APCC)* (pp. 279-284). IEEE.

Feeney, L. M., & Nilsson, M. (2001). Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (Vol. 3, pp. 1548-1557). IEEE.

Gerla, M., & Tsai, J. T. C. (1995). Multicluster, mobile, multimedia radio network. *Wireless networks*, *1*(3), 255-265.

Hajami, A., Oudidi, K., & ElKoutbi, M. (2010, May). An enhanced algorithm for MANET clustering based on multi hops and network density. In *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on* (pp. 181-188). IEEE.

Hakimi, S. L. (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations research*, *12*(3), 450-459.

Heinzelman, W. B., Chandrakasan, A. P., & Balakrishnan, H. (2002). An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on wireless communications*, *1*(4), 660-670.

Kariv, O., & Hakimi, S. L. (1979). An algorithmic approach to network location problems. II: The p-medians. *SIAM Journal on Applied Mathematics*, *37*(3), 539-560.

Kim, H., Seok, Y., Choi, N., Choi, Y., & Kwon, T. (2005, January). Optimal multi-sink positioning and energy-efficient routing in wireless sensor networks. In *International Conference on Information Networking* (pp. 264-274). Springer, Berlin, Heidelberg.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436-444.

Lochin, E., Fladenmuller, A., Moulin, J. Y., Fdida, S., & Manet, A. (2003). Energy consumption models for ad-hoc mobile terminals. In *In Med-Hoc Net*.

Mishra, K. K., & Harit, S. (2010). A fast algorithm for finding the non-dominated set in multi-objective optimization. *International Journal of Computer Applications*, *1*(25), 3539.

Mladenović, N., Brimberg, J., Hansen, P., & Moreno-Pérez, J. A. (2007). The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, *179*(3), 927-939.

Mohammed, N., Otrok, H., Wang, L., Debbabi, M., & Bhattacharya, P. (2011). Mechanism design-based secure leader election model for intrusion detection in MANET. *Dependable and Secure Computing, IEEE Transactions on*, *8*(1), 89-103.

Peiravi, A., Mashhadi, H. R., & Hamed Javadi, S. (2013). An optimal energy - efficient clustering method in wireless sensor networks using multi - objective genetic algorithm. *International Journal of Communication Systems*, *26*(1), 114-126.

Pencheva, T., Atanassov, K., & Shannon, A. (2009). Modelling of a roulette wheel selection operator in genetic algorithms using generalized nets. *Int. J. Bioautomation*, *13*(4), 257-264.

Rajan, M. A., Chandra, M. G., Reddy, L. C., & Hiremath, P. (2008). Concepts of graph theory relevant to ad-hoc networks. *Int. J. of Computers, Communications & Control*, *3*, 465-469.

Reese, J. (2006). Solution methods for the p-median problem: An annotated bibliography. *Networks*, *48*(3), 125-142.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, *20*, 53-65.

Safa, H., Artail, H., & Tabet, D. (2010). A cluster-based trust-aware routing protocol for mobile ad hoc networks. *Wireless Networks*, *16*(4), 969-984.

Sett, S., & Thakurta, P. K. G. (2015, March). Effect of optimal cluster head placement in MANET through multi objective GA. In *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in* (pp. 832-837). IEEE.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, *4*(2), 65-85.

Wu, J., Dai, F., Gao, M., & Stojmenovic, I. (2002). On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. Journal of communications and networks, 4(1), 59-70.

Wu, W., Cao, J., & Raynal, M. (2009). Eventual clusterer: A modular approach to designing hierarchical consensus protocols in manets. *Parallel and Distributed Systems, IEEE Transactions on*, *20*(6), 753-765.

Younis, O., & Fahmy, S. (2004). HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. IEEE Transactions on mobile computing, 3(4), 366-379.

Younis, O., Krunz, M., & Ramasubramanian, S. (2006). Node clustering in wireless sensor networks: Recent developments and deployment challenges. *IEEE network*, *20*(3), 20-25.

Zhang, Y., Ng, J. M., & Low, C. P. (2009). A distributed group mobility adaptive clustering algorithm for mobile ad hoc networks. *Computer Communications*,*32*(1), 189-202.

Zhou, L., & Haas, Z. J. (1999). Securing ad hoc networks. *Network, IEEE*, *13*(6), 24-30.

Appendix A

List of Symbols

B: Set of member nodes

BFS: Breadth-First Search

BLUE: Subgraph consisting of the cluster heads, connections between cluster heads, and connections from cluster heads to sinks

BLUE(): Function that returns true if the *BLUE* graph is feasible or false otherwise

CH: Set of cluster heads

ConvNet: Convolutional Neural Network

cp: crossover point

CPU: Central Processing Unit

$d(v_i, v_j)$: Function that returns the Euclidean distance between nodes v_i and v_j

E: Set of edges on an undirected graph

e: The maintenance energy used at a cluster head

E_{recv}: Energy used by a node receiving a message

E_{send}: Energy used by a node sending a message

$f(E_{recv}, CH)$: Function representing the energy usage of a network

FS: Fitness value

FS_{max}: Maximum fitness value in a population of chromosomes

FS_{min}: Minimum fitness value in a population of chromosomes

G(V.E): An undirected graph with vertices *G(V.E)* and edges *E*

GA: Genetic Algorithm

g_i: Gene at index *i* in a chromosome

GPU: Graphics Processing Unit

J: joule

k: Factor representing the extra energy required in sending versus receiving messages by nodes

MA_{blue}: Binary m by (m + c) matrix where m = number of cluster heads and c = number of sinks, used to simplify calculations of *BLUE* graph feasibility

$MA_{chromosome}$: Binary matrix representing a chromosome

$MA_{Connect}$: Function returning 1 if an edge exists between two nodes and 0 if it does not

$MA_{frequency}$: 1 by n matrix where n is the number of nodes, recorded the frequency of appearance in shortest-path trees for each node

MANET: Mobile Ad-hoc NETwork

MA_{red}: m by (n-m) matrix where n= number of nodes in the network and m = number of cluster heads and the summed columns indicated if a node was in range of a cluster head

MAXFREQ(): Function that returns the node with the highest value in $MA_{frequency}$

MINFREQ(): Function that returns the node with the lowest value in $MA_{frequency}$

numC: Number of chromosomes in a population

P_i: Probability of selection of chromosome *i*

RED: Subgraph consisting of the member nodes and connections from member nodes to cluster heads

RED(): Function that returns true if the *RED* graph is feasible or false otherwise

S: Set of sinks

S_{comp}: Connected component of a sink

TF(): The target function to be minimized by the algorithm

V: Set of vertices on an undirected graph

WSN: Wireless Sensor Network

x_i: The number of messages sent at node v_i

y_i: The number of messages received at node v_i

Appendix B

As noted in the research it was not technically feasible to check every possible configuration in order to determine the optimal fitness value of the large networks. Considering a 100-node network, there are 2^{100} or 1.27×10^{30} possible network configurations. Checking one trillion possible configurations per second, this would take several times over the age of the universe to test every one. Only smaller networks could have every configuration tested to find the optimal. This approach of testing every possible solution is often referred to as the brute-force method. Therefore methods were developed in order to estimate the optimal fitness values of the larger networks used in this research based on optimal results found using the brute-force approach on smaller networks. The three large networks were the 44-node, the 100-node, and the 100-node-sparse networks.

Three methods were developed and are detailed below. The first involved using brute-force results from small networks to prove that with regular node placement, the optimal fitness value of the whole network could be obtained by finding the brute-force value for half the network and doubling it. The second method built on the results of the first. It showed that points plotted on a graph, using the number of nodes as the x-axis and the fitness value as the y-axis, would define a curve used to predict fitness values for larger numbers of nodes. Again the brute-force method was used to determine fitness values for networks from 10 through 25 nodes. The third method involved estimating the number for hops from each node to the sink. An equation was developed and tested against the values found in small networks (up to 25 nodes) using the brute force method.

Developing the estimate for the 44-node network was straightforward due the regular placement of the nodes. Recall the structure of the 44-node network.

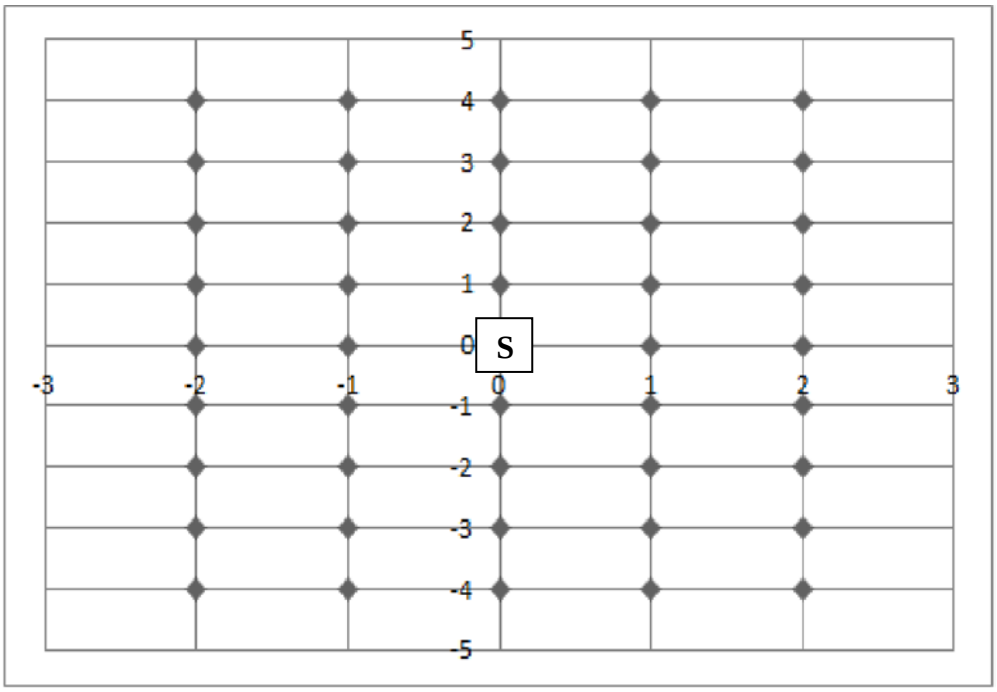

Figure B1. 44-node Network

The fitness value was estimated by taking exactly half of the nodes and finding the optimal fitness value through brute-force, and then doubling that value for the network with twice the nodes. The 22-mode network used for this test is shown below.

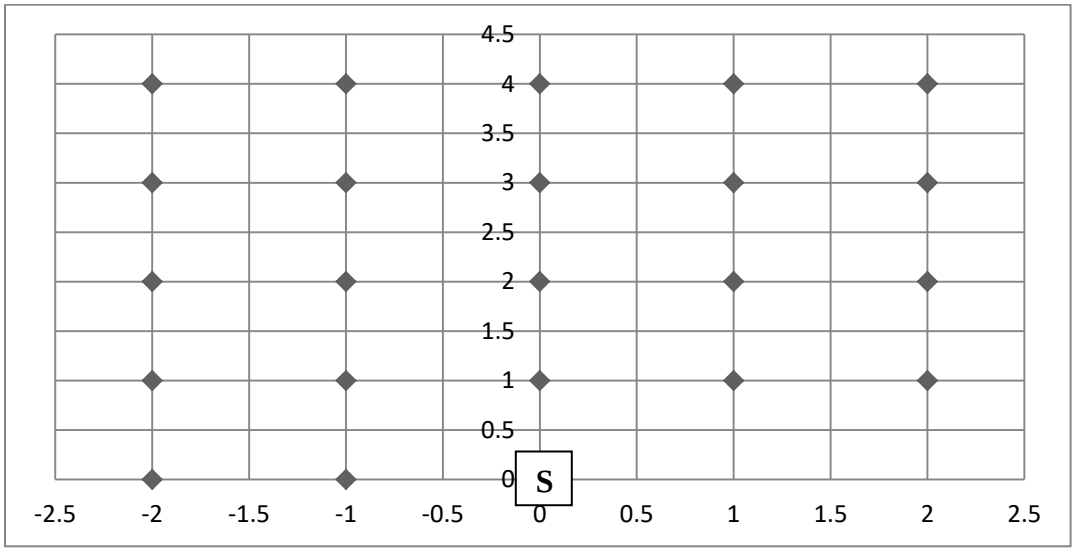

Figure B2. 22-node Network

The 22-node network yielded a fitness value of 98, making the estimate of the 44-node network optimal 196. This idea was further confirmed by testing the 7-node network and 14-node network as well as a 12-node and 24-node with similar configurations as the 22 and 44-node networks above. The results that were obtained using the brute-force method are shown below in Table B1.

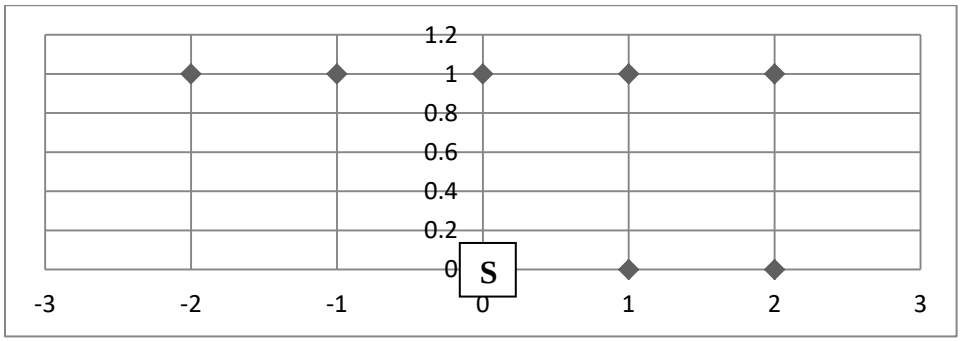

Figure B3. 7-node Network

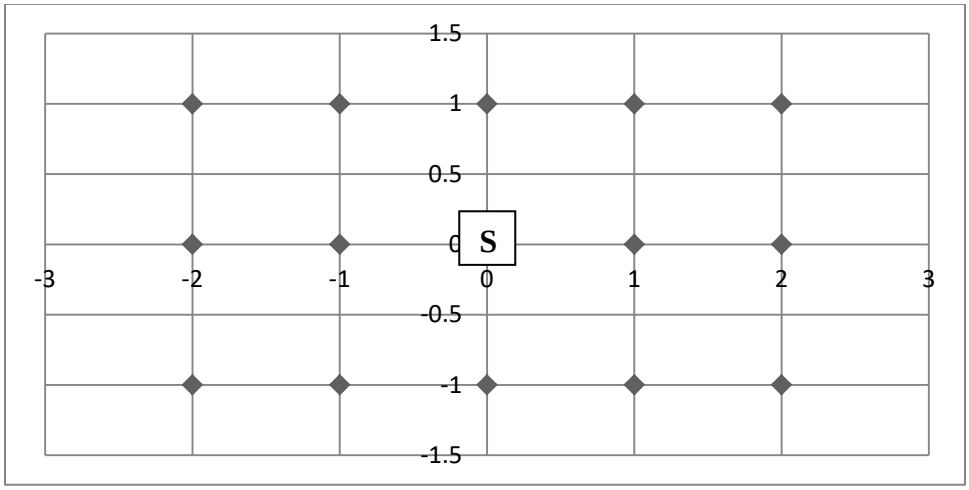

Figure B4. 14-node Network

	7-node	14-node	12-node	24-node
Fitness value	19	38	34	68

Table B1.

As shown, the hypothesis of using half of the network and doubling the value for the whole network was shown to be valid.

The two 100-node networks did not have a regular structure of node placement therefore a different method had to be implemented. The fitness values obtained from the whole networks (14-node, 24-node, 38-node, and 44-node) were plotted on a graph and a trend line fitted as shown below.

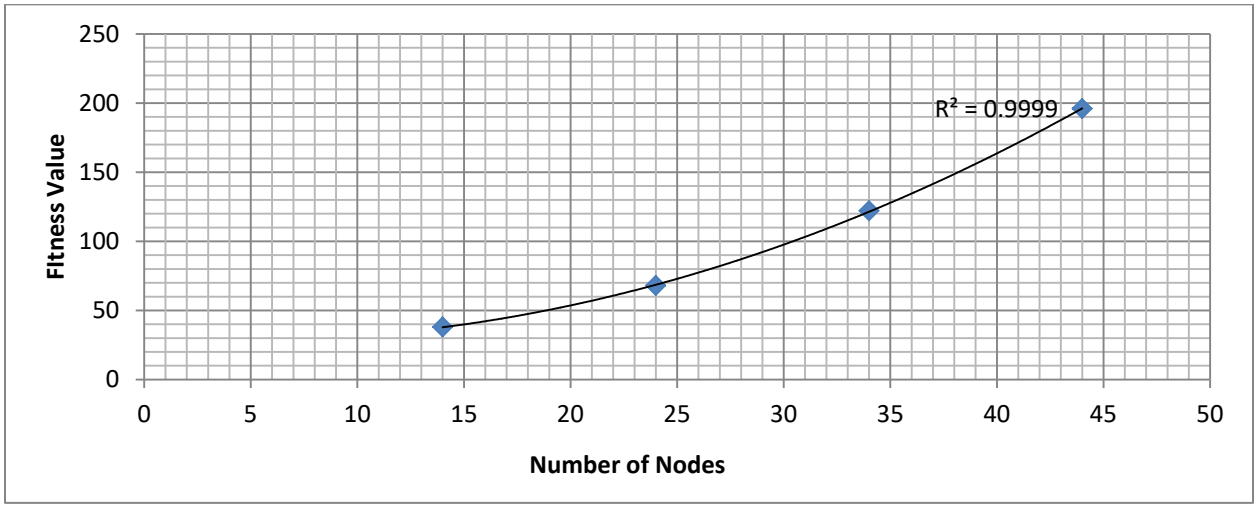

Figure B5. 44-node Trend Line

As shown, a trend line could be drawn through the points that represented a good fit for all of the values. If the last value was removed from the graph, it could be seen that the trend line provided a reasonable estimate of the value as shown below in Figure B6.

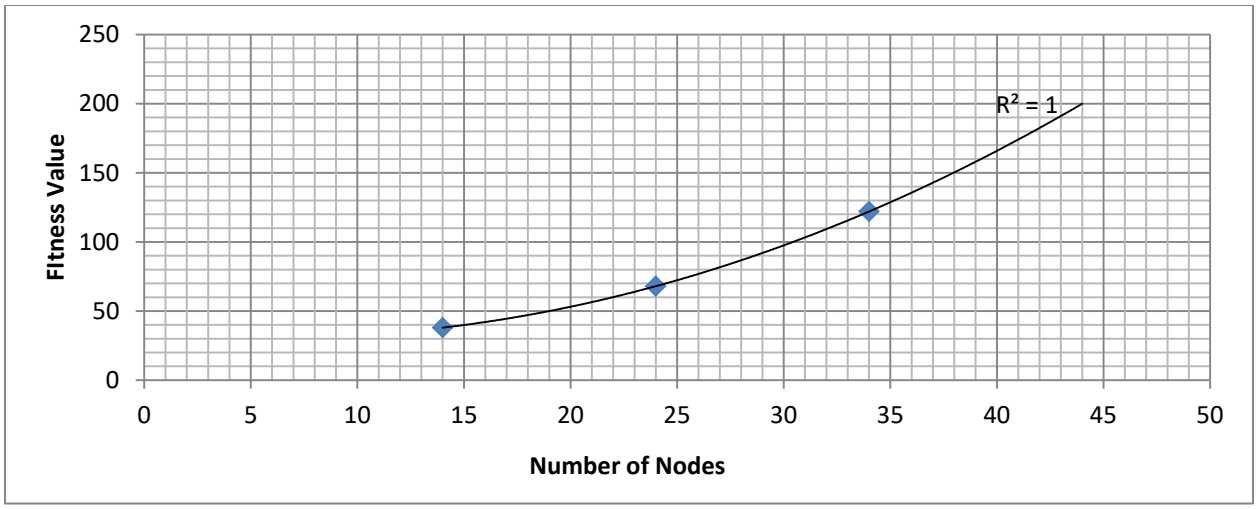

Figure B6. Estimating the Last Value

Using this method for the two 100-node networks yielded the results shown below. The plotted values were obtained through sorting the nodes by distance from the sink and then removing all but the 25 nearest nodes from the original networks and then using the brute-force method to determine the optimal fitness value. This process was repeated for 6 additional data points; 22, 20,17,15,12, and 10 nodes, using the brute-force method for all with both 100-node networks.

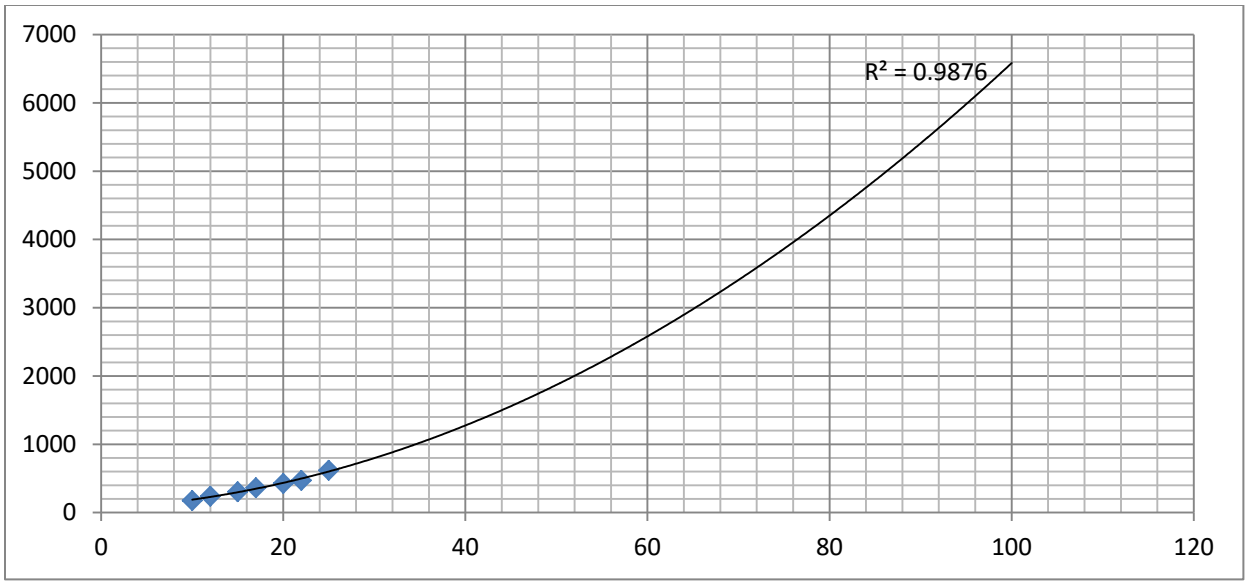

Figure B7. 100-node, Predicted Optimal Value

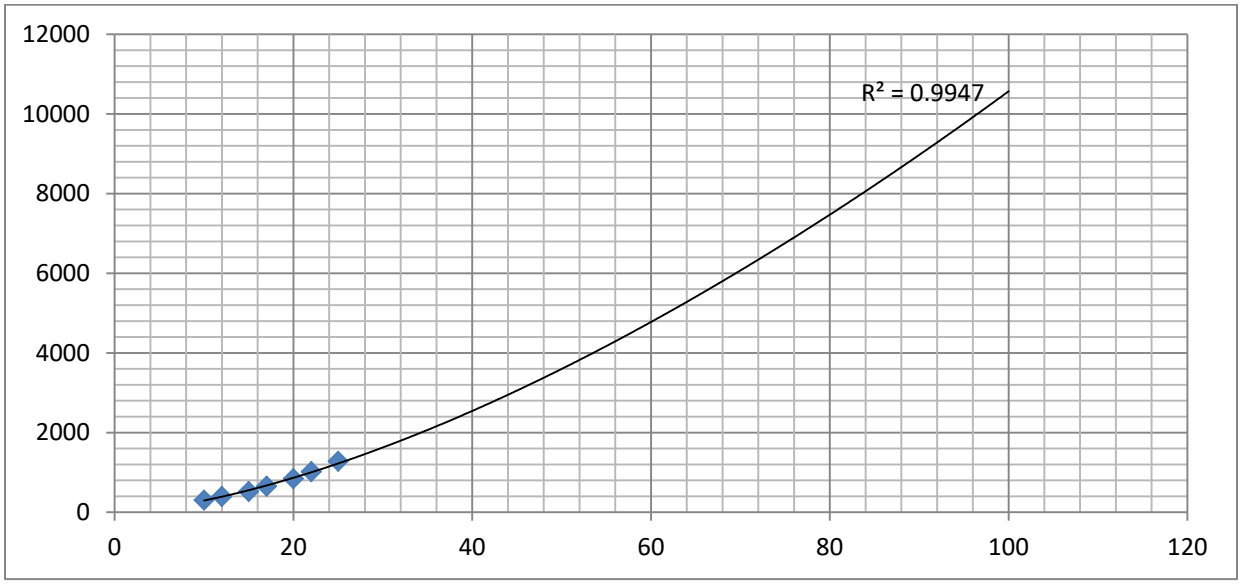

Figure B8. 100-node-Sparse, Predicted Optimal Value

The 44-node network line (Figure B6) provided a better fit than the 100-node networks (Figures B7, B8). However if a greater power polynomial was used for the trend lines it appeared to over fit the lines, resulting in a predicted fitness value many thousands above the

values observed in the research. Therefore the lines shown in the graph were used to estimate the optimal values for the 100-node networks. The graphs resulted in estimated optimal values of 6,582 for the 100-node network and 10,658 for the 100-node sparse network.

The methods shown above provided reasonable estimates of optimal values for the large networks. Along with the methods shown, as additional validity, the 25-node test network was run against the repair algorithms and the best values found through the repair algorithms exactly matched the optimal value found through the brute-force method as shown below.

	Optimal (Brute Force)	**BFSrepair**	**ARrepair**
Fitness Value	1280	1280	1280

Table B2. 25-node Network results

While the methods shown above appear successful, it was found during testing that due to the distance on the curve from the data points to the predicted value that small changes in the points could create relatively large swings in the predicted value. There was also a concern that this approach may not be as reliable for the networks with random node placement versus the regular node placement. Therefore an additional estimation method was implemented for the two 100-node networks.

The third estimation method functioned through calculating the energy required to send each message to the sink. For each node the distance to the sink was calculated. Then the number of hops to the sink was estimated based on the transmission range. The rationale behind the equation used is shown below.

As shown in Figure B9 below, if there exists a sufficient density of intermediate nodes between the node being evaluated and the sink, then the number of hops to the sink would equal the distance divided by the transmission range (Tr).

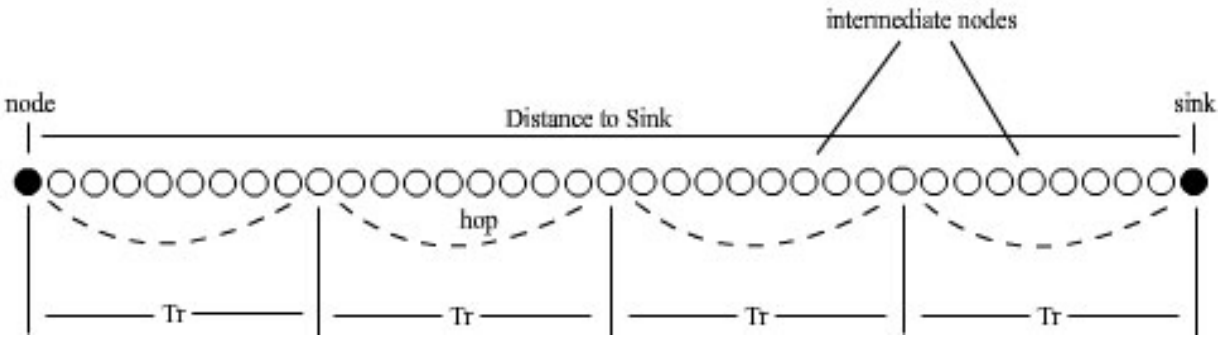

Figure B9.

As shown the equation below is valid for this condition.

NumHops = DistanceToSink/Tr

However, in an actual network the density of the nodes will be lower and therefore the number of hops will increase based on the density of the nodes. This is shown in Figure B10 below.

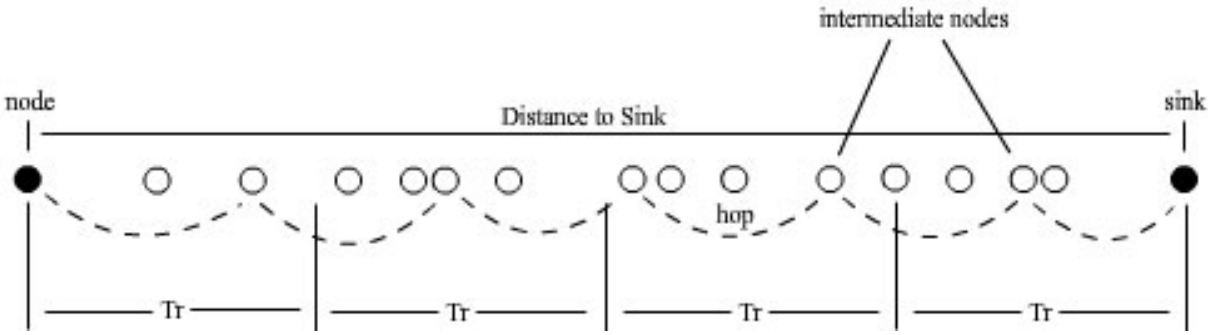

Figure B10.

As shown a factor based on the density of the nodes was required in order to create a reliable estimation. This factor was called the Density Factor and is included in the equation below.

NumHops = DensityFactor * (DistanceToSink/Tr)

In an infinitely dense network the density factor would be 1, and will increase as the there are fewer intermediate nodes in the network. The factor was determined experimentally to be 1.36 for the two 100-node networks. Meaning that for every length of transmission range there were 1.36 hops in the network. The Number of hops was converted to an integer and rounded down. Additionally nodes where NumHops = 0 had the value of NumHops changed to 0.3. This was to account for the situation where there were multiple nodes within one hop of the sink. Not all of them would be cluster heads and therefore there would be an additional hop for some of the nodes. This value was found experimentally to be 0.3. This approach was tested on the same smaller networks from 10 to 22 nodes and the values compared to the optimal values obtained via the brute-force method.

With the sparse 100-node network the estimated values averaged to be within 2.6% of the optimal values and with the standard 100-node network the estimated values were within 4.3% of the optimal values. The results are shown in Table B3 below.

	100-Node Standard			100-Node Sparse		
#Nodes	**Optimal**	**Estimated**	**%Difference**	**Optimal**	**Estimated**	**%Difference**
10	176.3	161	8.7%	306.05	276	9.8%
12	240	234	2.5%	396.35	390	1.6%
15	304.95	309	1.0%	523.5	539	3.0%
17	364.1	378	3.8%	654.4	653	0.2%
20	426.15	452	6.0%	842.5	824	2.2%
22	471.65	507	7.5%	1021.95	1033	1.1%
25	619.45	623	0.5%	1280.45	1286	0.4%
		Average	4.3%		**Average**	2.6%

Table B3. Estimated and Optimal Values

Using the above method on the two 100-node networks yielded estimated optimal values of 6,752 for the 100-node and 10,323 for the 100-node sparse network. These values are near those obtained with the earlier graphing method. Since it was not possible to calculate the optimal for these networks, the average of the two methods was used in the research as the estimated optimal, 6,667 for the 100-node network and 10,490 for the 100-node sparse network.