

AN ADAPTIVE CLUSTERING ALGORITHM  
FOR WIRELESS SENSOR NETWORKS

By

ALIREZA BOLOORCHI TABRIZI

Bachelor of Science in Computer Engineering

Amirkabir University of Technology

Tehran, Iran

2008

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
December 2011

AN ADAPTIVE CLUSTERING ALGORITHM  
FOR WIRELESS SENSOR NETWORKS

Thesis Approved:

Dr. M. H. Samadzadeh

---

Thesis Adviser

Dr. Johnson Thomas

---

Dr. Nazanin Rahnavard

---

Dr. Mark E. Payton

---

Dean of the Graduate College

## PREFACE

In the context of Wireless Sensor Networks (WSN), the interactions of the sensors in order to provide service to a specific job is a significant issue. Services provided by WSNs include collecting data from the environment and aggregating them to address queries, or processing the collected data and using the result to adjust a number of environmental parameters such as temperature and moisture. In WSNs, a number of nodes may need to be used by the same job simultaneously and these nodes usually need to interact with one another. A number of studies have been conducted to minimize the overhead of such interactions and to optimize the energy and other resources utilized in the process. It is desirable in WSNs that enough resources be assigned to the jobs in such a way that the jobs can acquire their needed resources as fast as possible. The distances among the sensors that are assigned to carry out a specific job typically constitute an important factor in saving energy in the context of communications among these sensors.

This thesis concerned introducing a clustering algorithm to enhance the efficiency of resource assignment by reducing the distances among the cooperating sensors. Furthermore, in the proposed algorithm, clusters are formed with different sizes in order to be able to assign just enough number of sensors to a requested job. The sizes of the clusters in the network were determined based on an input to the algorithm that

contained the initial number of required clusters of each size. The algorithm forms clusters in such a way that the clusters' sizes are adapted to the input for the network for the purpose of serving the incoming jobs better. Upon providing a different input, the algorithm adaptively changes the network's clusters.

In this thesis, a basic version of the clustering algorithm was developed. In the simulation results, several issues were recognized and resolved by revising/extending the initial version of the algorithm. In a second extension, the algorithm was complimented by attaching the isolated nodes (i.e., nodes not assigned to any cluster) to the clusters that did not obtain a sufficient number of nodes. The clustering algorithm was executed with a pseudo-randomly generated input. The results showed that the number of clusters with each size matched the requirements for the given input. The algorithm was also run with 100 different pseudo-randomly generated inputs and rectangular-grid networks of different sizes.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
1.1 Background and Problem Statement.....	1
2.1 Assumptions and Research Objectives .....	2
II. REVIEW OF LITERATURE .....	3
2.1 Chapter overview .....	3
2.2 Wireless Sensor Networks (WSN).....	3
2.3 Clustering Algorithms .....	4
2.4 Four Popular Clustering Algorithms for WSNs .....	5
2.4.1 K-mean .....	5
2.4.2 Low Energy Adaptive Clustering Hierarchy (LEACH) .....	6
2.4.3 Hybrid Energy-Efficient Distributed Clustering (HEED).....	7
2.4.4 An Energy Efficient Hierarchical Clustering Algorithm for WSNs .....	8
III. NETWORK MODEL AND ASSUMPTIONS .....	10
3.1 Chapter Overview .....	10
3.2 Network Model.....	10
3.3 Assumptions .....	11
IV. PROPOSED CLUSTERING ALGORITHM'S DESIGN.....	15
4.1 Chapter Overview .....	15
4.2 Basic Algorithm: Clusterheads and the Advertisement Protocol .....	15
4.3 Extension 1: Re-Selection of the Clusterheads.....	17
4.4 Extension 2: Assigning Isolated Nodes to Clusters .....	19

Chapter	Page
V. ANALYSIS AND SIMULATION .....	21
5.1 Chapter Overview .....	21
5.2 Introduction .....	21
5.3 Simulation Results for the Basic Algorithm.....	24
5.4 Simulation Results for Extension 1.....	26
5.5 Simulation Results for Extension 2.....	27
5.6 Simulation Results for Different RoAs' Distributions Among the Clusterheads .....	29
5.7 Scalability Analysis.....	32
5.7.1 Number of Nodes .....	33
VI. SUMMARY AND FUTURE WORK .....	34
6.1 Summary .....	35
6.2 Future Work.....	36
REFERENCES.....	37
APPENDICES.....	39
Appendix A .....	40
Appendix B .....	42

## LIST OF TABLES

Table	Page
Table 1. Number of Clusters of each cluster size or type .....	12
Table 2. The map of Table 1 to cluster types with a granularity factor of 4 .....	13
Table3. An example of calculating the average number of clusters for each cluster sizes .....	24

## LIST OF FIGURES

Figure	Page
Figure 1. Result of initial clusterhead selection. The darker nodes are pseudo-randomly chosen clusterheads and the other nodes represent the regular nodes before being assigned to a cluster.....	22
Figure 2. An example of the distribution of the Range of Advertisements (RoA) among clusterheads. RoAs define clusters sizes. ....	23
Figure 3. Comparison of the expected RoAs and the average number of clusters with each size in the results for 100 executions of the basic algorithm on the same distribution of RoAs.....	25
Figure 4. Visual simulation result of the basic algorithm. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes in each cluster.....	25
Figure 5. Comparison of the expected RoAs and the average number of clusters with each cluster size in the results for 100 executions of the extension 1 algorithm using the same distribution of RoAs.....	26
Figure 6. Visual simulation result of extension 1. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes in each cluster.. ....	27
Figure 7. Comparison of the expected RoAs and the average number of clusters with each size in the simulation results for 100 executions of extension 2 algorithm on the same distribution of RoAs. ....	28
Figure 8. Visual simulation results of extension 2 (crawling method), (a) choice 1 (b) choice 2. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes of each cluster.....	29
Figure 9. Results for execution of the proposed algorithm for 100 pseudo-randomly generated distributions.....	31
Figure 10. Results for several network sizes: (a) $10 \times 10$ nodes (b) $10 \times 20$ nodes (c) $30 \times 30$ nodes. The nodes of a cluster have the same color and shape, and	



the clusterheads are the larger nodes with the same color and shape as the  
regular nodes of each cluster ..... 32

## CHAPTER I

### INTRODUCTION

#### 1.1 Background and Problem Statement

In Wireless Sensor Networks (WSNs), resource management and energy efficiency are important issues in the context of node interactions [Heinzelman et al. 00] [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07] [Zhang et al. 06].

It is necessary to assign sensors to the services that WSNs provide for incoming queries in a way that each service acquires its required sensors in a finite amount of time. A service could be monitoring environmental parameters such as temperature and moisture in different sections of a forest, or doing intrusion detection in different sections of a bank. A sensor might need to interact with a number of other sensors of the network in order to obtain the results of their operations for serving a specific query. Communication might be also between the sensors and a center that manages the sensors or functions as a sink to which the processed, collected, or generated data are sent for further processing.

In WSNs, if some of the nodes are not needed for any requested service, a procedure is required to ask the nodes to go to sleep in order to save energy [van Dam and Langendoen 03]. Matching the number of nodes in a cluster with the number of nodes

needed to provide a service, to which the cluster is going to be assigned, could be a good solution for efficient communication and resource management. The reason is that the nodes that are awake and serving a requested service could interact with less energy consumption since they are in the same cluster.

## 1.2 Assumptions and Research Objectives

In this thesis work, a clustering algorithm was provided to group the sensors of a network into clusters of different sizes. The nearest sensor to the geographical center of a cluster is called a clusterhead [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07]. The proposed clustering algorithm aimed to decrease the sum of the distances between the nodes and the clusterhead in each cluster. To this end, an iterative approach was used to find best clusterheads on the basis of their locations. To test the proposed clustering algorithm, a list that shows how many clusters with each size are expected to exist in the clustered network was used. The list was generated pseudo-randomly in this thesis.

## CHAPTER II

### REVIEW OF LITERATURE

#### 2.1 Chapter overview

Resource management provided by clustering algorithms for the purpose of energy efficiency in WSNs has been the area of research for several studies during the last decade. In this chapter, a brief background for WSNs and clustering algorithms is provided. Four of the most notable clustering algorithms for WSNs are also discussed with the aim of using some of their best practices in designing the proposed clustering algorithm.

#### 2.2 Wireless Sensor Networks (WSNs)

The emergence of low-power wireless sensors is the result of recent advances in Micro-Electro-Mechanical Systems or MEMS base sensor technology and low power electronics [Dong et al. 97] [Clare et al. 99] [Chandrakasan et al. 99]. Sensors are typically capable of processing data and communication. Wireless Sensor Networks collect data from the environment, process them in some cases, and provide access to the data. A sensor in a WSN could measure a number of its environment's parameters (e.g., temperature and sound) and convert the collected data into electrical

signals. Processing of the signals generated by a sensor reveals information about the environment around that sensor [Abbasi and Younis 07]. Sensors are generally weaker than the nodes in mobile ad hoc networks in that they are less mobile, more limited in capabilities, and more densely deployed [Younis and Fahmy 04].

### 2.3 Clustering Algorithms

Backer and Jain [Backer and Jain 81] defined clustering analysis as follows: “a group of objects is split into a number of more or less homogeneous subgroups on the basis of an often subjectively chosen measure of similarity such that the similarity among the objects within a subgroup is larger than the similarity among the objects belonging to different subgroups”.

Clustering or categorizing entities with similarities into groups has been used in computer science (web mining, computer vision, machine learning, and wireless sensor networks), in life sciences and medical sciences (genetics, biology, microbiology, psychiatry, and pathology), in social sciences (sociology, psychology, archaeology, and education), in earth sciences (geography, geology, and remote sensing), and in economy (marketing and business) [Xu and Wunsch 05].

The idea of clustering has been used in a number of studies on WSNs to enhance energy efficiency [Heinzelman et al. 00] [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07]. One of the frequently-considered constraints in designing new clustering algorithms is that a clustering algorithm for WSNs should provide energy efficiency while using the maximum capability of the nodes in the respective clusters to give the best possible performance.

In addition to the regular sensing roles in WSNs, some nodes serve another role. These nodes, one per cluster, are referred to as clusterheads and usually are in charge of communicating with other clusterheads, base stations, and of course other nodes in the corresponding cluster [Abbasi and Younis 07]. Base stations are the sink nodes to which the transferred data from the network are destined for further processing, and they usually have higher computational and communicational capabilities. Clusterheads sometimes have management responsibilities as well such as assigning less energy consuming tasks to nodes with less energy, dealing with node failures, and controlling intra-cluster communication and traffic [Bandyopadhyay and Coyle 03] [Crosby and Pissinou 07].

## 2.4 Four Popular Clustering Algorithms for WSNs

### 2.4.1 K-mean

The distance between nodes and their clusterheads is an important factor in the context of Wireless Sensor Networks where energy constraints limit the communication capabilities [Abbasi and Younis 07] [Heinzelman et al. 00]. Several existing clustering algorithms such as K-mean [McQueen 66] create clusters based on the distance between pairs of nodes in a network. In the K-mean algorithm, which aims to divide a WSN into  $k$  clusters,  $k$  sensors are chosen randomly as clusterheads. Sensors join the clusters with the geographically nearest clusterhead to them. After all nodes join the cluster, the first step is completed. Then the nearest nodes to the geographical center of each cluster are chosen as the new clusterheads. The same procedure is repeated until the clusterheads do not change anymore [McQueen 66].

#### 2.4.2 Low Energy Adaptive Clustering Hierarchy (LEACH)

One of the popular clustering algorithms for Wireless Sensor Networks is Low Energy Adaptive Clustering Hierarchy (LEACH) [Heinzelman et al. 00]. In LEACH, all sensors in the network are homogeneous and energy constrained. They assume a radio model for the transmitters and receivers. In this radio model, the receivers' energy consumption depends on the message size, and the transmitters' energy consumption depends on the message size and the square of the distance the data being transmitting. In LEACH, it is also assumed that the radio channels are symmetric in the sense that for a given SNR, the energy required to transmit a message from node A to node B is the same as the energy required to transmit a message from node B to node A. It is also assumed that all the sensors are sensing with a same rate so they always have data to send to the end users.

Sensors become clusterheads based on two parameters. The first parameter is a suggested percentage for clusterheads in the network that is given as an input to the algorithm. The second parameter is the number of times a node has been a clusterhead. Clusterheads advertise or broadcast their status as clusterheads in the network. The strengths of the signals that are used for communication in the network decrease as the signals move away from the source. Based on the strengths of the signals that the sensors receive from the clusterheads, the non-clusterhead nodes join a cluster with the clusterhead that can be reached with the least energy consumption for communication [Heinzelman et al. 00].

In LEACH, clusterheads are changed in a timely manner to prevent them from running out of energy which could come about much earlier than the other nodes. This

could be considered the distinguishing feature of LEACH, a conventional clustering algorithm that has fixed clusterheads during its lifetime (i.e., the time period before the first sensor dies as a result of energy depletion). The LEACH protocol has been compared to three other protocols, namely, direct transmission, minimum transmission control, and static clustering. In the direct transmission protocol, each sensor node transmits directly to the sink, and it is efficient when there is a small coverage area and/or high receive cost. Traffic is routed through independent nodes in minimum transmission energy protocol which is a good solution when the average transmission distance is large. In static clustering, the nodes in each cluster transmit the collected data to the clusterhead and the clusterhead transmits it to the sink.

It has been shown that compared to conventional clustering algorithms, Heinzelman et al.'s algorithm increases network lifetime [Heinzelman et al. 00].

#### 2.4.3 Hybrid Energy-Efficient Distributed Clustering (HEED)

Another clustering algorithm for Wireless Sensor Networks is Hybrid Energy-Efficient Distributed clustering (HEED) [Younis and Fahmy 04]. Younis and Fahmy assumed that the sensors are stationary and all have the same amounts of energy and identical processing capabilities. Analogous to LEACH, the aim of the HEED protocol is prolonging network lifetime by adaptively changing the clusterheads based on their energy. In their protocol, Younis and Fahmy used a second parameter for decision making in the situations when a node receives advertisements from more than one clusterhead. HEED works based on the probability of two clusterheads being in each other's transmission range, i.e., the probability of existence of nodes that might receive clusterhead advertisement from both clusterheads. The smaller this probability is, the



more uniformly distributed the clusterheads are in the network, as Younis and Fahmy demonstrated. They modified LEACH slightly in order to be able to compare its result in terms of network lifetime with their algorithm's results. They showed that their algorithm outperforms this extension of LEACH. Younis and Fahmy showed that the improvement is a result of a better choice of clusterheads by HEED as compared to LEACH where clusterheads are chosen randomly.

#### 2.4.4 An Energy Efficient Hierarchical Clustering Algorithm for WSNs

Bandyopadhyay and Coyle [Bandyopadhyay and Coyle 03] introduced an energy efficient clustering algorithm for WSNs in which sensors in the network join clusters based on each sensor's distance from the clusterheads. In this algorithm, a probabilistic approach is used to select the clusterheads. The event of a node becoming a clusterhead follows a binomial distribution where each node becomes a clusterhead with probability  $p$  that is determined based on the required number of clusters in the network, and is assumed to be provided as an input to their algorithm [Bandyopadhyay and Coyle 03].

In Bandyopadhyay and Coyle's work, each clusterhead advertises itself to other nodes in the network. These advertisements are broadcast in the network with the range of no more than a specific number of  $k$  hops. The number of hops is the number of intermediate nodes in the path from a clusterhead to a node that receives the advertisement.

Bandyopadhyay and Coyle tried to find optimal values for parameters  $p$  and  $k$  to minimize the energy used in the network [Bandyopadhyay and Coyle 03]. They simulated their algorithm and provided a comparison with another clustering algorithm, namely, Max-Min D-Cluster [Amis et al. 00]. In the Max-Min D-Cluster algorithm,

networks are clustered in such a way that each node is either a clusterhead or at most  $d$  hops away from a clusterhead with  $d \geq 1$ .

A hierarchical clustering algorithm was also provided by Bandyopadhyay and Coyle [Bandyopadhyay and Coyle 03]. In this algorithm, after a network is clustered, which is the first level of a hierarchy, the clusterheads are considered as nodes of a new network. The new network is clustered as the second level of the hierarchy, and again the clusterheads of the second level build a new network of their own. This procedure continues up to a given level (the optimized number of levels is provided by Bandyopadhyay and Coyle) [Bandyopadhyay and Coyle 03]. In the hierarchical clustering algorithm, the first level would be the same as the basic protocol where each node elects itself as a clusterhead with probability  $p_1$  and the clusterheads advertise themselves within the range of  $k_1$  hops. For subsequent levels, the clusterheads of the previous level elect themselves as the clusterheads of a new level  $i$  with probability  $p_i$  and advertise themselves within the range of at most  $k_i$  hops. Bandyopadhyay and Coyle also introduced a method to optimize the values of  $p_i$  and  $k_i$  in level  $i$  clusters in an attempt to provide an energy efficient scheme [Bandyopadhyay and Coyle 03].

## CHAPTER III

### NETWORK MODEL AND ASSUMPTIONS

#### 3.1 Chapter Overview

In this chapter, a network model and several assumptions about the characteristics of the network to which the new proposed clustering algorithm applies are specified. First, a number of network characteristics and assumptions are discussed. The assumptions are tightened in Section 3.3.

#### 3.2 Network Model

In this thesis work, the focus was on a network with a number of homogeneous components where each component is a processing unit with local memory, and no limitation is imposed on the number of components in the network. The components are deployed in rows and columns, and the layout of the network is assumed to be rectangular with hard boundary (i.e., a rectangular grid). The components to the right and left of a component as well as above and below it are called its neighbors. Each component is directly connected to its four neighbors. The distance between two neighbors is defined to be one unit. Since the layout has hard boundaries, there are

components at the periphery of the network which might have fewer than four neighbors. Investigation of the case where the components are distributed in a pattern more general than a rectangular array is relegated to the future work in this area.

The proposed clustering algorithm needs to be initially deployed with a centralized approach, i.e., a centralized entity is needed to manage the clustering.

The components of the network are to be divided into several clusters of not-necessarily-equal sizes. One of the components in each cluster is used as a clusterhead to *aggregate* the output of the components of the cluster and to manage the communication among the components in the cluster. Aggregation here refers to any process to which the output of the components of a cluster could be subjected, e.g., concatenation of the outputs or removal of the possible overlaps of the outputs.

### 3.3 Assumptions

The input to the proposed clustering algorithm is a list of the number of clusters with different sizes, i.e., number of nodes. An example of the input list is given in Table 1. In this example, it is assumed that the clusters have between 6 and 25 nodes.

Depending on the network that uses the proposed clustering algorithm, the input list can be generated using predictive methods based on historical data, e.g., based on the number of clusters of different sizes in the past. However, the focus of this work is on the clustering algorithm. Therefore, as a simplifying assumption and to maintain narrow research focus, the input based on which the algorithm forms the clustered network is generated pseudo-randomly. To evaluate the level of dependence of the algorithm on the pseudo-randomly generated input, the algorithm was tested for 100 different pseudo-randomly generated inputs as outlined in Section 5.6.

Table 2. Number of clusters of each “cluster size” or “type”.

Cluster size	Number of Clusters	Cluster size	Number of Clusters
6	1	16	1
7	2	17	1
8	0	18	0
9	1	19	0
10	2	20	2
11	0	21	3
12	1	22	0
13	2	23	3
14	0	24	2
15	4	25	0

As for using historical data to predict the needed number of clusters with each size, generating an accurate table like Table 1 might not be easy. In other words, determining the number of clusters needed for each cluster size precisely would be a nontrivial task. Without the loss of generality, the clusters with sizes that are only slightly different could be put into one group as a cluster type resulting in a number of cluster types. The granularity of this grouping can be decided based on the accuracy of the method used for generating the input list. In the example provided in Table 2, the clusters in Table 1 are grouped in a way that the difference between any pair of cluster sizes in a cluster type is at most 4 nodes, referred to as the granularity factor for the rest of this thesis report. For example, the number of clusters for the first cluster type is equal to the sum of the number of clusters with each size in that cluster type which are 1, 2, 0, and 1 in Table 1, and 4 for the first cluster type in Table 2. It is obvious that Table 1 is a special case of Table 2 with a granularity factor of 1.

Table 2. The mapping of Table 1 to cluster types with a granularity factor of 4.

Cluster size	Number of Clusters		Cluster type	Cluster Type's Range	Median of the Group range	Number of Clusters
6	1	⇒	1	{6, 7, 8, 9}	7.5	4
7	2					
8	0					
9	1					
10	2	⇒	2	{10, 11, 12, 13}	11.5	5
11	0					
12	1					
13	2					
14	0					
15	4	⇒	3	{14, 15, 16, 17}	15.5	6
16	1					
17	1					
18	0					
19	0					
20	2	⇒	4	{18, 19, 20, 21}	19.5	5
21	3					
22	0					
23	3					
24	2					
25	0	⇒	5	{22, 23, 24, 25}	23.5	5

A parameter is assigned to each clusterhead that shows the number of nodes that the corresponding cluster needs. This parameter is named the Range of Advertisement (RoA) which shows the expected cluster size. The namesake for the RoA is that the number of

nodes that are joining a cluster should be proportional to the distance (range) up to which the related clusterhead advertises the membership message.

All clusters in a cluster type were assigned the same RoA which is the median of the cluster type's range. For example, the RoA of all five clusters of the second cluster type in Table 2 is equal to 11.5. A method could be used to lessen the effect of the inaccuracy that may be introduced due to the grouping method proposed earlier in this section. Instead of assigning the same RoAs to all clusters in a cluster type, the clusters in a specific cluster type could be distributed among the other clusters with sizes in the cluster types' ranges. For example, the number of clusters in the second cluster type in Table 2 is 5. Instead of assigning 5 clusters with an RoA of 11.5, there could be 1.25 clusters with RoA of 10, 1.25 clusters with RoA of 11, 1.25 clusters with RoA of 12, and 1.25 clusters with RoA of 13. Although the number of clusters should be a natural number as one would expect, the real number that is mentioned here shows the *expected* number of clusters for each RoA. To make it clearer, assume that the number of clusters to which each RoA in a cluster type's range are assigned are chosen using a pseudo-random number generator that is based on a uniform distribution.

The computational complexity of the proposed clustering algorithm is an issue to be considered. Based on the studies on a special case of the proposed clustering algorithm (k-mean) [McQueen 66], the problem is computationally difficult (NP-hard). With different assumptions, the questions of best case and worst case complexities of the algorithm could be investigated.

## CHAPTER IV

### PROPOSED CLUSTERING ALGORITHM'S DESIGN

#### 4.1 Chapter Overview

Characteristics of the input to the proposed algorithm were presented in Chapter III. In this chapter, first a basic version of the proposed clustering algorithm is explained. The basic algorithm is based on an advertisement protocol. The proposed clustering algorithm is then improved with two extensions. The first extension consists of an iterative advertisement protocol with the purpose of improving the clustering process. The second extension introduces a method for assigning the nodes which have not already joined a cluster to different clusters.

#### 5.2 Basic Algorithm: Clusterheads and the Advertisement Protocol

In the proposed algorithm, each node can be a clusterhead with a probability of  $P_{ch}$ . Considering a node becoming a clusterhead as “success” and not becoming a clusterhead as “failure” where the event of a node becoming a clusterhead is independent of other nodes, the number of clusterheads in the network follows a binomial distribution.

$$P_{ch} = \frac{N_{ch}}{N} \quad (7)$$

where  $N_{ch}$  is the number of clusterheads and  $N$  is the number of nodes in the network.



$N_{ch}$  clusterheads are chosen from among the nodes in the network pseudo-randomly. Each clusterhead advertises membership messages which are messages that ask network nodes to join the clusters of the advertising clusterheads. The membership messages are sent to each clusterheads' directly-connected neighbors (four nodes). The nodes that are not clusterheads and receive the advertisement, join the cluster. If a node receives two or more membership messages, it joins the cluster for which the difference between its number of acquired nodes and the respective clusterhead's RoA is larger. In the next step, the recently-joined nodes advertise membership messages to their own directly-connected neighbors. Ideally, this procedure should continue until every cluster acquires exactly a number of nodes that is equal to its RoA. However, based on simulations (Section 6.3), it was determined that the basic algorithm has a number of drawbacks as listed below.

1. There might be a number of nodes that are neither clusterheads nor members of a cluster, to be referred to hereafter as isolated nodes.
2. There might be a number of clusters in close proximity to each other that cannot grow and accumulate the sufficient number of nodes, i.e., reach their assigned RoAs, the reason being that the clusterheads have been chosen pseudo-randomly from the nodes and are not *distributed evenly* in the network to be able to acquire their required number of nodes. The clusterheads are distributed evenly in a network if they are distributed evenly in a way that all of them are able to acquire their required nodes while staying at the geographical center of their cluster. Uneven distribution of the clusterheads may also cause the creation of *entangled isolated nodes* which are nodes that are not connected to any cluster and are

trapped among some clusters that have already accumulated a sufficient number of nodes.

3. Some clusters may have constrained surroundings because either there are other clusterheads as their neighbors or they are next to the boundary of the rectangular grid network. Since a rectangular layout with hard boundary is assumed for the network (see Chapter III), there may be a number of clusterheads close to the periphery that cannot grow in one or even two directions. Moreover, from the simulations, it has been observed that as expected the nodes near the boundary are more likely to stay isolated.

In the following sections, two extensions to the proposed basic clustering algorithm are introduced with the purpose of addressing the above three issues.

### 5.3 Extension 1: Re-Selection of the Clusterheads

In this extension, the nodes that have been selected as clusterheads by the clustering algorithm are released and the *central node* of each cluster is chosen as a future clusterhead. If the network is assumed as a two dimensional Cartesian coordinate system, the central node of each cluster is the node whose coordinates are closest to the average of the coordinates of all the nodes in the cluster in both  $x$  and  $y$  axes.

The extended proposed algorithm basically consists of iterations of the basic algorithm followed. The iterations terminate once the clusterheads do not change their locations anymore, i.e., when the sum of the distances of the current clusterheads and the previous ones in each cluster approaches  $0$ . Note that  $0$  is a limit value and, since a discrete environment is assumed, there may be a case where two or more nodes are qualified for being clusterheads in a cluster and selection of a clusterhead may switch

back and forth among those nodes continually. Therefore, once the locations are changing less than a threshold value, to be referred to as the *convergence threshold*, the process is stopped. The convergence threshold is a system parameter and its value is decided based on a trade-off between the algorithms convergence time and the even distribution of the clusterheads.

This extension should provide a better distribution of the clusterheads since it is designed to separate the clusterheads that are undesirably close to each other. The modified clustering algorithm also places the clusterheads away from the boundary based on the number of nodes in each cluster.

Since the clusterheads are distributed pseudo-randomly in the network, it is possible, in spite of the re-selection of the clusterheads toward the center of clusters, that a number of the clusterheads still get entangled among some other clusterheads and not be able to acquire all of their required nodes. This problem is addressed by pseudo-randomly selecting another node in the network as the clusterhead instead of any of the clusterheads that have not reached a threshold number of nodes in the previous iteration. The RoA of the current clusterhead should be assigned to the new one. The threshold value for the number of nodes that each cluster should acquire in each iteration was chosen based on the RoA of that cluster. The closer threshold values to the RoAs of the clusterheads resulted in fewer isolated nodes in the basic algorithm and a larger convergence time.

Simulation results showed that the problem that a number of nodes may not be members of any cluster still exists. The next extension addresses this issue.

#### 5.4 Extension 2: Assigning Isolated Nodes to Clusters

In this extension, two possible alternative improvements are proposed and their comparison is discussed in the analysis and simulation chapter (Section 5.5) in more detail.

In the first improvement, before each iteration of re-selecting the clusterheads, the isolated nodes are assigned to the nearest clusterhead. In the situations where there are more than one clusterhead with the same minimum distance to the node, the clusterhead that needs more nodes to reach its specified population size is chosen.

An alternative improvement is introduced in this thesis. This improvement, to be referred to as the *crawling method*, attaches the isolated nodes to the clusterheads that need more nodes to reach their RoAs, i.e. the neediest clusters. The isolated node should not be added directly to the neediest cluster because it prevents the protocol from guaranteeing nearest distances between nodes and their corresponding clusterheads. Another algorithm was proposed to help the isolated nodes crawl in the network to reach the neediest cluster. The algorithm is outlined below.

1. Each isolated node, marked as the current node, finds a shortest path through the network to the neediest cluster.
2. In the shortest path, the current node joins the cluster that the next node belongs to.
3. If the next node in the shortest path is not a member of the neediest cluster, set the next node as the current node and go to Step 2.
4. If there is no other isolated node in the network, exit, else go to Step 1.

After the algorithm runs, the clusterhead re-selection is re-applied and the new clusterheads will be the nodes in the center of the clusters with the inclusion of the new additions.

Since the clusterheads are initially selected pseudo-randomly, in some cases the algorithm may not converge to minimize the sum of the distances, i.e., the sum of the distances may just keep changing. To deal with this problem, a threshold value was used for the number of iterations. If within that threshold number of iterations the algorithm did not converge, the clusterheads are re-selected pseudo-randomly, and the algorithm starts anew. The threshold value could be chosen based on different factors such as the number of nodes, historical data, etc.

## CHAPTER V

### ANALYSIS AND SIMULATION

#### 5.1 Chapter Overview

The design of the proposed clustering algorithm was explained in Chapter IV. The proposed clustering algorithm's implementation and evaluation are described in the current chapter. First, the basic algorithm is evaluated and, based on the observed drawbacks, the first extension is added. The results of the extension are compared with the basic algorithm. Next the second extension is implemented and added to the proposed algorithm, and the results are analyzed.

The proposed clustering algorithm was tested for different number of nodes in the network and 100 different pseudo-randomly generated inputs (Section 5.6). The inputs contain information about the number of clusters of certain sizes. The results suggest that the proposed clustering algorithm's performance is independent of network size and specific input.

#### 5.2 Introduction

As it is mentioned in Chapter III, the network has a rectangular layout with hard

boundary, and the clusterheads are chosen based on the computed probabilities as given in Section 4.2.

Figure 1 shows a  $20 \times 20$  network after choosing the clusterheads. The nodes with darker colors are the clusterheads and the rest are regular nodes. The input could be considered as the distribution of the RoAs (see Section 3.3) among the clusters in the network.

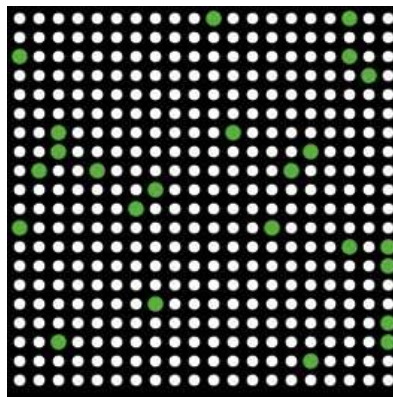


Figure 1. Result of initial clusterhead selection. The darker nodes are pseudo-randomly chosen clusterheads and the other nodes represent the regular nodes before being assigned to a cluster.

In this section, the *RoAs' distribution* among the clusterheads is assumed to be as given in Figure 2. The given RoAs' distribution is consistently used for all simulation results in this chapter. In the example of this chapter, five cluster types with sizes between 5 and 29 are considered and the granularity factor is 5.

In Figure 2, the number of clusters for each RoA is not a whole number and the reason is explained in Section 3.3. The results of using the algorithm for 100 different pseudo-randomly generated RoAs' distributions among the clusterheads are discussed later in the thesis (Section 5.6).

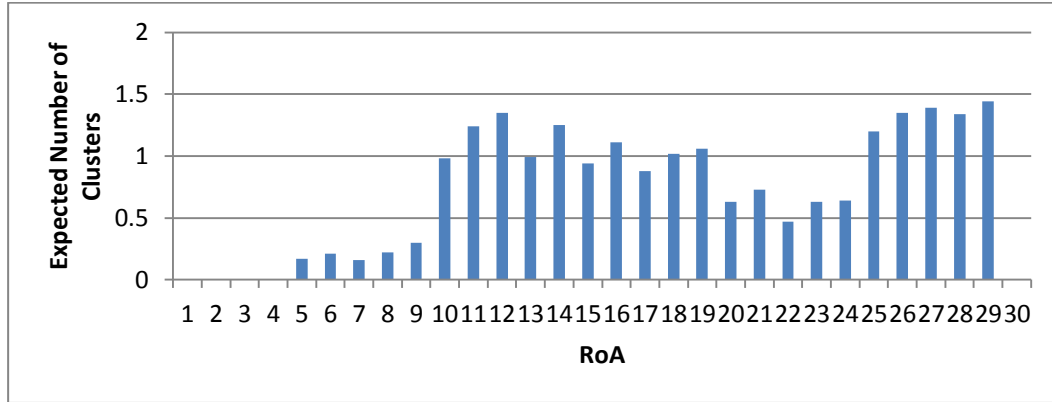


Figure 2. An example of the distribution of the Range of Advertisements (RoA) among clusterheads. RoAs define clusters sizes.


An advertisement protocol was used to assign the nodes to clusters. All clusterheads advertise to take members with the same priority. To this end, the clusterheads are asked to advertise one step at a time, i.e., first to their neighbors, next to the neighbors of their neighbors, and so on. The process of stepwise advertising ceases when all clusterheads obtain their required number of nodes or there are no nodes that respond to the advertisements. Based on the simulations in this thesis, it has been observed that it is harder for the clusters with larger RoAs to acquire sufficient number of nodes. Thus, another choice could be giving the clusters with larger RoAs higher priority in taking nodes. To this end, the clusterheads could be prioritized in a non-ascending order of their RoAs in order to give the clusterheads with larger RoAs more opportunity to acquire nodes.

To make the results that are provided in this chapter more reliable, the clustering algorithm was executed *100* times and the average of the result of each execution was used for analysis. Table 3 describes how the average of 3 execution of clustering algorithm could be generated. In this example, 5 clusters are considered in a cluster type with the range of {6, 7, 8, 9}.



Table3. An example of calculating the average number of clusters for each cluster size.

Cluster ID	1	2	3	4	5
Cluster size after 1 <sup>st</sup> execution	7	7	6	7	9
Cluster size after 2 <sup>nd</sup> execution	8	7	9	9	6
Cluster size after 3 <sup>rd</sup> execution	9	7	7	7	9



Cluster Sizes	6	7	8	9
Number of clusters after 1 <sup>st</sup> execution	1	3	0	1
Number of clusters after 2 <sup>nd</sup> execution	1	1	1	2
Number of clusters after 3 <sup>rd</sup> execution	0	3	0	2
Average number of clusters	0.6	2.3	0.3	1.6

### 5.3 Simulation Results for the Basic Algorithm

The result of the execution of the basic algorithm is depicted in Figure 3. In the basic algorithm, only the advertisement procedure is executed and neither re-selection of the clusters is applied (extension 1) nor the isolated nodes are taken care of (extension 2). As it is observable in Figure 3, the number of clusters of sizes 18 and more (except 21) are less than the expected number. The number of clusters of sizes smaller than 17 are more than their expected number. The reason is that a number of the clusters do not acquire sufficient number of nodes, which is the result of the existence of the isolated nodes.

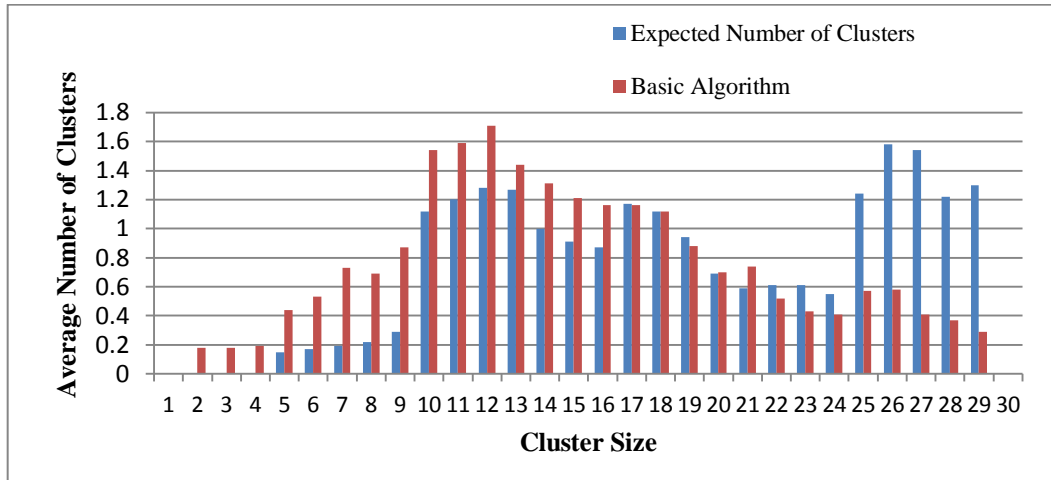


Figure 3. Comparison of the expected RoAs and the average number of clusters with each size in the results for 100 executions of the basic algorithm on the same distribution of RoAs.

The clusterheads are distributed pseudo-randomly therefore a number of them might prevent other clusterheads from acquiring nodes and, a number of the clusterheads might fall on the boundary and not be able to advertise in all directions.

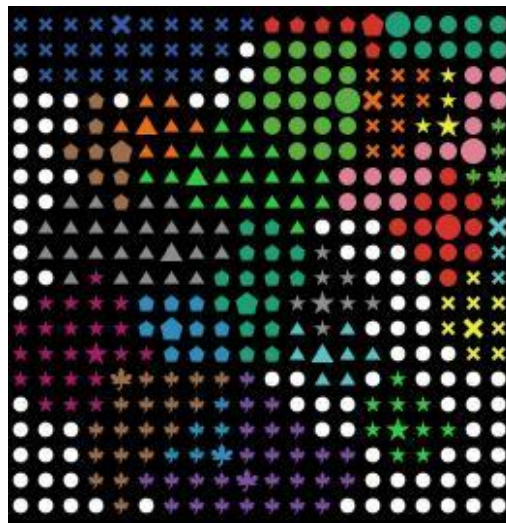


Figure 4. Visual result of the basic algorithm. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes in each cluster.

In the example depicted in Figure 4, the distribution of the clusterheads in the network as a result of applying the basic algorithm indicates that several clusterheads may be undesirably close to one another, and a number of them might be on or too close to the boundary.

#### 5.4 Simulation Results for Extension 1

In this stage, a procedure for re-selecting clusterheads was added to the algorithm. Figure 5 shows the simulation result of adding the procedure for re-selecting the clusterheads and the results of iterations of the algorithm until the change in the clusterheads' locations became at most one unit for each clusterhead. One unit is the convergence threshold value that was discussed in Section 4.3.

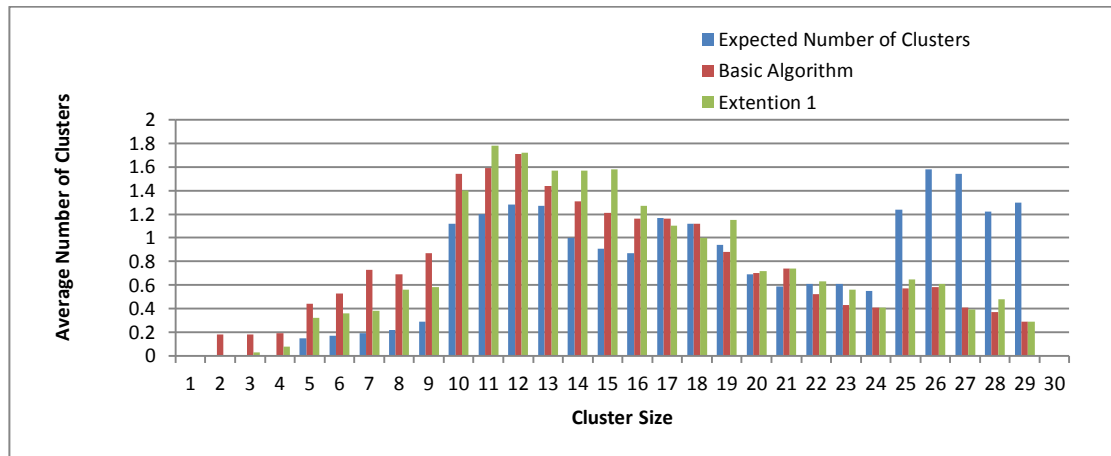


Figure 5. Comparison of the expected RoAs and the average number of clusters with each cluster size in the results for 100 executions of the extension 1 algorithm using the same distribution of RoAs.

The number of clusters with different sizes was still not matching the RoAs distribution and the clusterheads with higher RoAs still could not obtain a sufficient number of nodes although all clusterheads were far enough from the boundary and each

other to be able to reach their RoAs. The clusterheads were also close to the centers of their own clusters. Figure 6 depicts the visual simulation result of this extension.

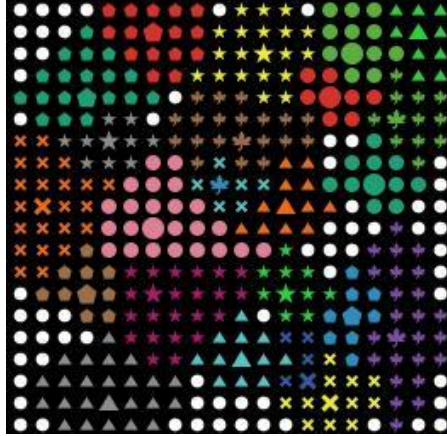


Figure 6. Visual simulation result of extension 1. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes in each cluster.

### 5.5 Simulation Results for Extension 2

To choose between the two proposed methods of attaching the isolated nodes to the clusters, both methods that were proposed in Section 4.4 were implemented and the results were compared. The distribution of the resulting clusters in the network was compared with the distribution of the RoAs among the clusterheads. It was found that the convergence time for the first method was less than the convergence time for the second method. However, the distribution of clusters with different sizes did not match the RoAs distribution among the clusterheads. A better distribution of the clusterheads seems to be necessary to be found using an optimization approach, e.g., by using genetic algorithms, which was not covered in this thesis.

There are two choices for attaching the isolated nodes to clusters using the crawling method as introduced in Section 4.4. One choice is to attach the isolated nodes before each re-selection of the clusterheads and the other choice is to execute the crawling method once right at the end of last iteration of reclustering. Simulations showed that the second choice worked better in that the clusterheads were distributed more evenly and the algorithm converged faster. The later convergence time of the first choice might be because of a number of nodes that were not directly attached to their clusters causing the crawling procedure to be executed several more times thus making the clusters unbalanced. As a result of unbalanced clusters, more iterations were needed in re-selecting the clusterheads in order to stabilize the location of the clusterheads. In the simulation runs, the convergences time for the first choice and the second choice were 33.73 and 2.34, respectively. The two numbers are the average number of iterations that each method needs to converge. They are the average number of iterations for 100 executions of the algorithm. The result for the crawling method is given in Figure 7. The visual results of the algorithm for the two choices are depicted in Figure 8.

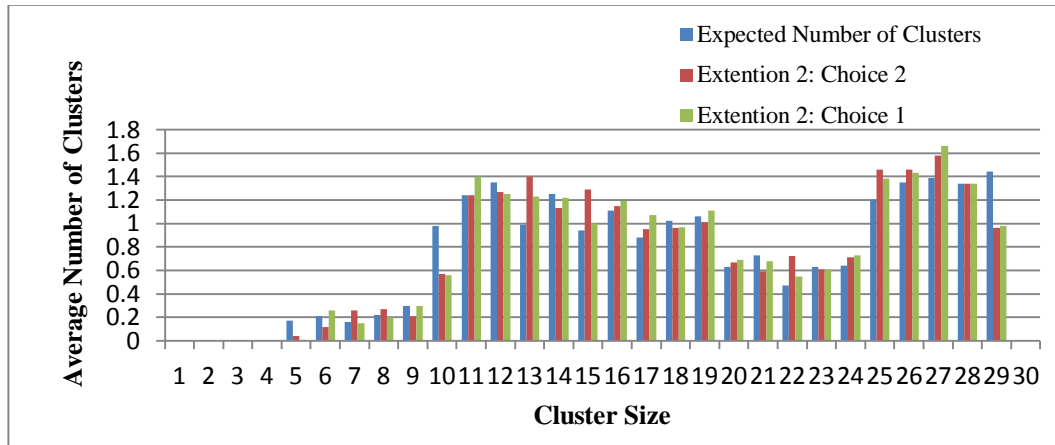


Figure 7. Comparison of the expected RoAs and the average number of clusters with each size in the simulation results for 100 executions of extension 2 algorithm on the same distribution of RoAs.

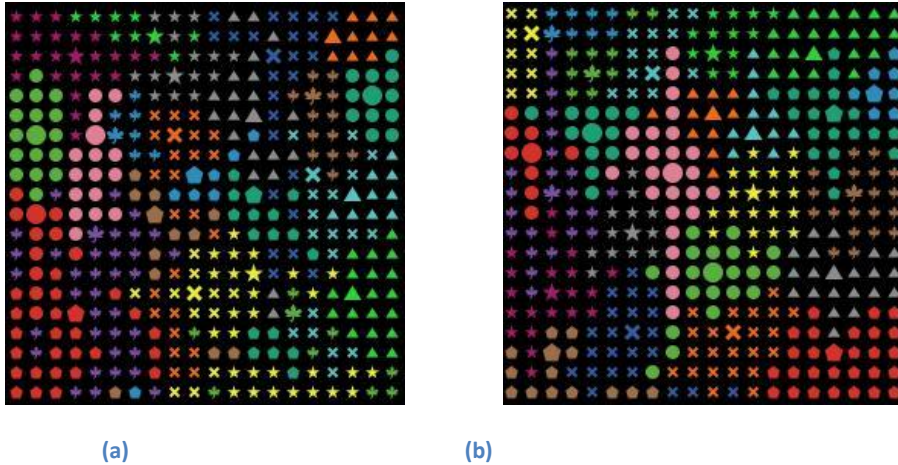


Figure 8. Visual simulation results of extension 2 (crawling method), (a) choice 1 (b) choice 2. The nodes of a cluster have the same color and shape, and the clusterheads are the bigger nodes with the same color and shape as the regular nodes of each cluster.

In Figure 7, compared to extension 1, the number of clusters with each size is closer to the expected number for that cluster size. Only in a few situations where the expected number of clusters changes from one cluster type to another was a large difference between the number of clusters and the expected number of clusters observed (e.g., from cluster size 9 to 10). Depending on the magnitude of change of the expected number of clusters in two adjacent cluster types, the first or the last cluster size in the cluster types' ranges might have more or fewer clusters than what was expected. This inequality was compensated for by other clusters in the cluster type. Since the clusters in one cluster type were supposed to have close cluster sizes, this inequality was considered negligible.

### 5.6 Simulation Results for Different RoAs' Distributions Among the Clusterheads

In the previous sections of this chapter, the simulation results presented were based on a specific distribution of the RoAs. In this subsection, it is shown that the algorithm worked well for 100 different pseudo-randomly generated distributions.

The difference between the number of the nodes that a cluster acquires and that cluster's RoA was calculated for all clusters. This number is referred to as the number of missing nodes of a cluster. Note that if the magnitude of the number of nodes acquired by a cluster is greater than its RoA, the magnitude of the number of missing nodes is considered to be a negative number equal to the difference between the number of acquired nodes and the RoA. The following formula calculates the sum of the differences in all clusters for the pseudo-randomly generated distributions.

$$d = \sum_{i=0}^{N_{ch}} |d_i| \quad (8)$$

In this formula,  $d$  is the sum of the differences,  $d_i$  is the number of missing nodes in cluster  $i$ ,  $|d_i|$  is the absolute value of  $d_i$  which is an integer between 0 and the RoA of the respective cluster, and  $N_{ch}$  is the total number of clusterheads in the network.

The ideal case for the result is for the sum of differences to be equal to 0 or the number of nodes in the clusters to be exactly equal to their RoAs.

The sum of the differences contains information about the total number of missing nodes in all clusters although it does not provide any information about how the missing nodes are distributed among the clusters. For example, consider the situation where  $N_{ch}$  is 10 and the total number of missing nodes in the clustered network is 20. It is possible that the number of nodes in cluster  $j$  be in the range  $[RoA_j - 2, RoA_j + 2]$ , where  $RoA_j$  is the RoA of cluster  $j$  with  $0 < j < N_{ch}$ . It is also possible that a cluster's missing nodes be 20 and any other cluster  $j$  have exactly  $RoA_j$  nodes. Formula (9) below, in addition to the total number of missing nodes, reflects another parameter that is related to how the missing nodes are distributed in the clusters of the network. In this formula, in each cluster more weight is given to the second missing node than the first one, more weight to

the third missing node than the second one, and so on for the rest of missing nodes. A geometric progression is utilized to reflect the weights.

$$d = \sum_{i=0}^{N_{ch}} a^{|d_i|} \quad (9)$$

In this formula, the value of  $a$  can be decided based on the criticality of not reaching RoAs in the clusters, i.e., not acquiring enough nodes. In the simulation runs reported in this work,  $a = 2$  was used. For instance, assume the situation where there are three clusters in the network and three nodes are missing. In situation where the three clusters miss one node each, the sum of the differences would be 3, but if one cluster misses three nodes and the others do not miss any node, the sum of the differences would be  $2^3$ .

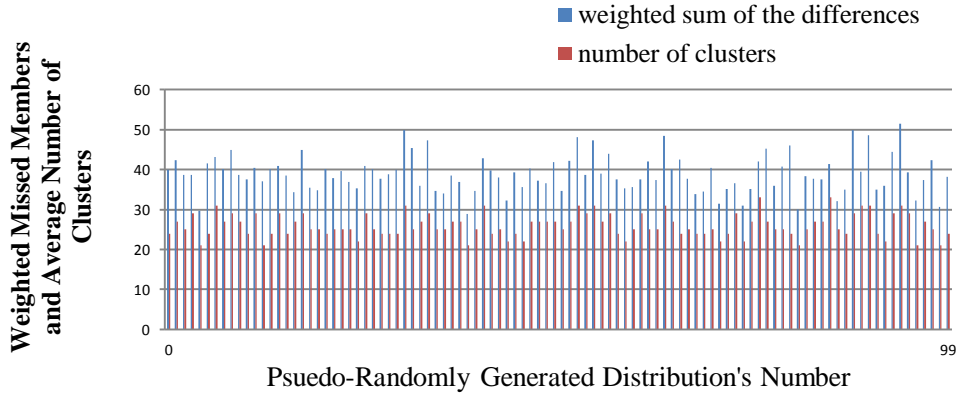


Figure 9. Results for execution of the proposed algorithm for 100 pseudo-randomly generated distributions.

Average simulation results for 100 executions of the proposed algorithm for any of the 100 pseudo-randomly generated distributions is shown in Figure 9. It appears that almost all the averages of the sum of the differences for all distributions in Figure 9 were between 30 and 50, and the number of clusters were in the range of 20 to 30. In the situation where all the missing nodes were from only one cluster, since the sum of the differences is at most 50, the average of the total number of missing nodes was less than



6 based on Formula 9 which is provided earlier in this section. If each cluster misses at most 2 nodes on average, the total of less than 25 nodes would be missing in the network. In both cases, the average result of executing the proposed algorithm was similar to the case study of this thesis, Sections 5.2 to 5.5.

### 5.7 Scalability Analysis

In the previous sections of Chapter V (Sections 5.2 to 5.6), a constant value of 400 nodes in 20 rows and 20 columns was assumed, and the number of cluster types was

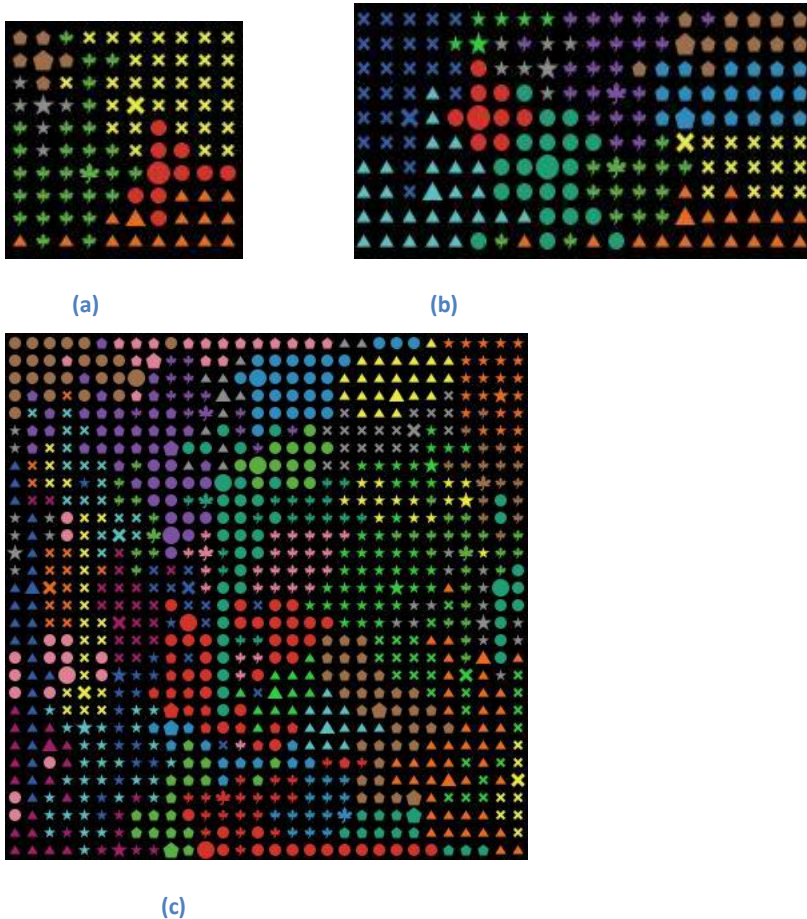


Figure 10. Results for several network sizes: (a)  $10 \times 10$  nodes (b)  $10 \times 20$  nodes (c)  $30 \times 30$  nodes. The nodes of a cluster have the same color and shape, and the clusterheads are the larger nodes with the same color and shape as the regular nodes of each cluster.

assumed to be 5. In the following section, the proposed algorithm is discussed for different values of number of nodes and cluster types.

#### 5.7.1 Number of Nodes

The algorithm was tested based on several choices for the number of rows and columns of the rectangular grid network. Results for three cases are shown in Figure 10.

## CHAPTER VI

### SUMMARY AND FUTURE WORK

#### 6.1 Summary

In Chapter I, the main objective of this thesis was presented. Chapter II provided background knowledge on Wireless Sensor Networks and clustering. The characteristics of four popular clustering algorithms were also discussed in Chapter II. Chapter III contained the description of the network model. The assumptions for this thesis were further described in Chapter III.

In Chapter IV, a new clustering algorithm was proposed to support Wireless Sensor Networks. The proposed algorithm groups the nodes of a given network into clusters of different sizes. The proposed clustering algorithm was built upon two quantitative aspects of the underlying network. The first aspect was the required number of nodes in each cluster that could be different for each cluster. The number of nodes in each cluster was aimed to match the size expected for that cluster. In this thesis, the expected size for each cluster was determined based on a pseudo-randomly generated input. The second aspect was the consideration of the sum of the distances between the nodes in each cluster and

its corresponding clusterhead. A clusterhead was chosen based on its location, which means that it was chosen in a way that the sum of the distances between nodes and the clusterhead was minimized. In this thesis, a basic algorithm was introduced and then extended in two stages.

The proposed algorithm was simulated based on pseudo-randomly generated expected number of clusters with different sizes. In Chapter V, the results of the simulation were provided. The results showed that the number of clusters with each size in the clustered network closely correspond to the given expected numbers. The proposed algorithm was executed based on 100 different pseudo-randomly generated expected number of clusters with different sizes to prototypically evaluate the applicability of the algorithm for more general cases. The scalability of the proposed algorithm was discussed in Section 5.7 and the observation is that the algorithm was working for different network sizes.

## 6.2 Future Work

The clustering algorithm that was introduced in this thesis was based on a number of assumptions. First, the proposed clustering algorithm is designed for Wireless Sensor Networks and it might be extendable for Network on Chips, cloud computing, and several other types of networks.

The network model was assumed to be in a rectangular layout with hard boundary. The sensors were considered to be placed in rows and columns and the distance between two neighbors was assumed to be one unit. The proposed clustering algorithm might be extended based on more generalized assumptions. The layout can be considered to be other geometric shapes or no specific geometric shape. The distribution of the sensors in

the network could be considered as not being in rows and columns or not even uniformly distributed. Instead of or in addition to the distance among the sensors, other aspects can be considered such as the cost of communication among the sensors.

## REFERENCES

- [Abbasi and Younis 07] A. A. Abbasi and M. Younis, "A Survey on Clustering Algorithms for Wireless Sensor Networks", *The Journal of Computer Communications*, Vol. 30, No. 14-15, pp. 2826-2841, October 2007.
- [Amis et al. 00] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, "Max-Min D-Cluster Formation in Wireless Ad Hoc Networks", *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, No. 1, pp. 32-41, Tel Aviv, Israel, March 2000.
- [Backer and Jain 81] E. Backer and A. K. Jain, "A Clustering Performance Measure Based on Fuzzy Set Decomposition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-3, No. 1, pp. 66-75, January 1981.
- [Bandyopadhyay and Coyle 03] S. Bandyopadhyay and E. Coyle, "An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks", *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communication (INFOCOM'03)*, Vol. 3, pp. 1713-1723, San Francisco, CA, April 2003.
- [Casella and Berger 90] George Casella and R. L. Berger, *Statistical Inference*, Duxbury Press, Pacific Grove, CA, 1990.
- [Chandrakasan et al. 99] A. Chandrakasan, R. Amirtharajah, Seonghwan Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, and A. Wang, "Design Considerations for Distributed Microsensor Systems", *Proceedings of IEEE 1999 Custom Integrated Circuits Conference (CICC)*, pp. 279-286, San Diego, CA, May 1999.
- [Clare et al. 99] L. P. Clare, G. J. Pottie, and J. R. Agre, "Self-Organizing Distributed Sensor Networks", *Proceedings of Society of Photo Optical Engineering (SPIE) Conference on Unattended Ground Sensor Technologies and Applications*, pp. 229-237, Orlando, FL, April 1999.

- [Crosby and Pissinou 07] G. V. Crosby and N. Pissinou, "Cluster-Based Reputation and Trust for Wireless Sensor Networks", *Proceedings of the Consumer Communications and Networking Conference (CCNC'07)*, pp. 604-608, Las Vegas, NV, January 2007.
- [Dong et al. 97] M. Dong, K. Yung, and W. Kaiser, "Low Power Signal Processing Architectures for Network Microsensors", *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 173-177, Monterey, CA, August 1997.
- [Heinzelman et al. 00] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", *Proceedings of the Hawaii International Conference on System Sciences (HICSS'00)*, Vol. 2, pp. 1-10, Wailea Maui, HI, January 2000.
- [Josang and Ismail 02] A. Josang and R. Ismail, "The Beta Reputation System", *Proceedings of the Fifteenth Bled Electronic Commerce Conference*, pp. 324-337, Bled, Slovenia, June 2002.
- [Keshavarzian et al. 06] A. Keshavarzian, H. Lee, and N. Venkatraman, "Wakeup Scheduling in Wireless Sensor Networks", *Proceedings of the Seventh ACM International Symposium on Mobile ad hoc Networking and Computing*, pp. 322-333, Florence, Italy, May 2006.
- [McQueen 66] J. B. McQueen, "Some Methods for Classification and Analysis of Multivariate Observations", *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1, pp. 281-297, Berkeley, CA, January 1966.
- [Pantazis et al. 09] N. A. Pantazis, D. J. Vergados, D. D. Vergados, and C. Douligeris, "Energy Efficiency in Wireless Sensor Networks Using Sleep Mode TDMA Scheduling", *The Journal of Ad Hoc Networks.*, Vol. 7, No. 2, pp. 322-343, March 2009.
- [Xu and Wunsch 05] R. Xu and D. Wunsch II, "Survey of Clustering Algorithms", *IEEE Transactions on Neural Networks*, Vol. 16, No. 3, pp. 645-678, May 2005.
- [Younis and Fahmy 04] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks", *IEEE Transactions on Mobile Computing*, Vol. 3, No. 4, pp. 366-379, October 2004.
- [Zhang et al. 06] W. Zhang, S. K. Das, and L. Yonghe, "A Trust Based Framework for Secure Data Aggregation in Wireless Sensor Networks", *Proceedings of the Third Conference on Sensor and Ad Hoc Communications and Networks (SECON'06)*, Vol. 1, pp. 60-69, Reston, VA, September 2006.

## APPENDICES



## APPENDIX A

### GLOSSARY

Base Station	The sink nodes to which the transferred data from the network are destined and usually have higher computational and communicational capabilities.
HEED	Hybrid Energy-Efficient Distributed clustering.
Heterogeneous Network	A network connecting different types of devices.
Homogeneous Network	A network connecting similar devices.
Inter-cluster Communication	Communication among the clusters.
Intra-cluster Communication	Communication among the nodes of a cluster.
LEACH	Low Energy Adaptive Clustering Hierarchy.
Mobile Ad hoc Network	A Mobile Ad hoc NETWORK (MANET), is a self-configuring infrastructureless network of mobile devices connected by wireless links.
MEMS	Micro-Electro-Mechanical Systems, very small mechanical devices working with electricity.
Netlogo	A multi-agent programmable modelling environment [Wilensky 99].

NoC	Network-on-Chips is a new approach for designing the communication subsystem of a System-on-Chips (SoC).
SNR	Signal to Noise Ratio (SNR) is a measure that compares the level of a desired signal to the level of background noise.
SoC	System-on-Chips refers to integrating all components of a computer or other electronic systems into a single integrated circuit (chip).
WSNs	Wireless Sensor Networks, networks that consist of distributed sensors to monitor physical or environmental conditions such as sound, temperature, and moisture.

## APPENDIX B

### SOURCE CODE

This appendix contains the source code for the simulation part of this thesis.

```
.....  
;; AN ADAPTIVE CLUSTERING ALGORITHM FOR WIRELESS SENSOR NETWORKS  
;; Author Alireza Boloorchi Tabrizi  
;; Comments start with a semicolon ‘;’  
.....  
  
extensions [array table]  
  
; Declaring global variables  
  
globals [distances mostLikelyRoA RoA numberOfNeededClusters expectedValues  
probabilityOfRoA cluster_heads turtlesArray level numberOfClusterHeads  
numberOfClusterTypes overallNumberOfNeededClusters normalizationFactor  
currentMainTurtle p x advertiseBlock clusterHeadsBlock iterations totalDistances  
AVGNOMembers NOMembers lastOne firstOne startingCluster central-turtle nearestCluster  
isolatedNode nearestTemp endd check firstNode lastNode enddd]  
  
;In Netlogo, nodes are called turtles and they contain attributes which are declared  
;here  
  
turtles-own [joined beingCH rangeOfAdvertise number edge clusterType member flag  
CHMembers]  
  
;=====
```

```
;;Steup is the main procedure in this program and the program starts from this  
;procedure.  
  
;=====
```

```
to setup  
  
Let counter 0 ;To count the number of iterations  
  
Let temp array:from-list n-values 6 [0]  
  
Let tempMem array:from-list n-values numberOfClusterHeads [0]
```

```

;After each iteration of the algorithm, the average of the number of members of the
;clusters are updated in the AVGNOMembers array.

set AVGNOMembers array:from-list n-values (ceiling ((granularityfactor *
numberOfClusterTypes) ) + 1) [0]

set AVGNOMembersROA array:from-list n-values (ceiling ((granularityfactor *
numberOfClusterTypes) ) + 1) [0] let tempFirst 0

set firstOne 0 2

repeat 1[ ; This loop is for executing the algorithm for different RoA's distributions

    set counter counter + 1

    set temp AVGNOMembers

    set tempFirst firstOne

    start

    set firstOne tempFirst

    set AVGNOMembers temp

;set members tempMem

    printNumberOfMembers ;A

    set firstOne 1

]

let m 0

repeat 100[ ;This loop is to obtain the average of the number of clusters with each
;size for all the iterations.

    array:set AVGNOMembers m array:item AVGNOMembers m / counter

Print array:item AVGNOMembers m

set m m + 1

]

set endddd true

end

```

```

;=====

; This procedure takes care of the initializations of the environment before the
;algorithm starts.

;=====

to start

  clear-all

  create-turtles 400 ; There are 400 nodes in the network

  let n 1

  let m 0

  repeat 400 [ ; Initializes the turtles which are the nodes in the network

    ask turtle m [set number m set color white set edge "null" set joined "null" set
beingCH "null" set member "null" set CHMembers "null"]

    set m ( m + 1)

  ]

  let temp 0

;*****Boundary: Assign the attributes of the nodes that are on the
          ;boundary of the network layout.

ask turtle 0 [set edge ("left-down") ]

while [ n < 19] [

  ask turtle n [set edge "left" ]

  set n (n + 1)

]

ask turtle 19 [set edge "left-up"]

set n ( 1 )

while [ n < 19] [

  set temp (n * 20)

  ask turtle temp [set edge "down" ]

  set n (n + 1)

]

set temp (20 * 19 )

```

```

ask turtle temp [set edge "right-down" ]

set n ( 2 )

while [ n < 20] [

  set temp (n * 20 - 1 )

  ask turtle temp [set edge "up" ]

  set n (n + 1)

]

set temp (399)

ask turtle temp [set edge "right-up" ]

set n ( 1 )

while [ n < 19] [

  set temp (380 + n)

  ask turtle temp [set edge "right" ]

  set n (n + 1)

]

;*****

ask turtles [set shape "dot"]

ask turtles [set size 1.5]

let i 0

let j 0

set temp 0

;setting the location of each node

while [ i < 20][

  while [j < 20][

    set temp (i * 20 + j)

    ask turtle temp [setxy (i * 1)- 10 (j * 1)- 10]

    set j (j + 1)

  ]

  set i (i + 1)

```

```

    set j ( 0)
]
set granularityFactor 5;
probablityOfRoA
let g 0
end

;=====
; This procedure determines the number of expected nodes in each cluster.
;=====

to probablityOfRoA
set numberOfClusterTypes 5
set RoA array:from-list n-values numberOfClusterTypes [0]
set numberOfNeededClusters array:from-list n-values numberOfClusterTypes [0]
set totalDistances 1
;clustertypes
array:set RoA 0 (1)
array:set RoA 1 (2)
array:set RoA 2 (3)
array:set RoA 3 (4)
array:set RoA 4 (5)

;The numberOfNeededClusters is set for each cluster. It is modified in another
;version of the program to be generated pseudo-randomly.

array:set numberOfNeededClusters 0 (10);
array:set numberOfNeededClusters 1 (97)
array:set numberOfNeededClusters 2 (15)
array:set numberOfNeededClusters 3 (51)
array:set numberOfNeededClusters 4 (70)

sortByNumberOfNeededClusters

computeExpectedValues

```

```

findNumberOfClusterHeads

clusterHeads

;computeProbabilities

set iterations 0

assignDistancesToClusterHeads

let m 0

let notFinished true

while [notFinished = true][

set notFinished false

while [m < numberOfClusterHeads][

if [rangeOfAdvertise] of array:item cluster_Heads m - [member] of
array:item cluster_Heads m > 3

[ set notFinished true

]

set m m + 1

]

clusterInitial

]

; The isolatedNodes1 is one of the optional procedures of the algorithm which might
; be enabled for comparisons. This procedure has two versions isolatedNodes and
; isolatedNodes1.

;isolatedNodes1

end

```



```

;=====
; The clusters are sorted by the number of needed clusters where the clusters with
;more needs could be given a higher priority.
;=====

to sortByNumberOfNeededClusters

let k 0

let i 0

let j 0

let temp 0

let tempArr array:from-list n-values numberOfClusterTypes[0]

set tempArr (RoA)

while [ i < numberOfClusterTypes] [
  set j (i)
  while [j < numberOfClusterTypes] [
    if (array:item numberOfNeededClusters i < array:item numberOfNeededClusters j) [
      set temp (array:item numberOfNeededClusters i)
      array:set numberOfNeededClusters i array:item numberOfNeededClusters j
      array:set numberOfNeededClusters j temp
      set temp (array:item RoA i)
      array:set RoA i array:item RoA j
      array:set RoA j temp
    ]
    set j (j + 1)
  ]
  set i ( i + 1)
]

end

```

```

;=====
; This assigns clusterheads pseudo-randomly.
;=====

to clusterHeads

  let i 0

  set cluster_heads array:from-list n-values numberOfClusterHeads [0]

  while [i < numberOfClusterHeads] [

    array:set cluster_heads i one-of turtles

    ask array:item cluster_heads i [set member 1 ]

    if [beingCH] of array:item cluster_heads i != 1[

      ask array:item cluster_heads i [set beingCH 1 set size 2 ]

      set i (i + 1)

    ]

  ]

; print cluster_heads

end

;=====

; This procedure is for future work and uses a method to estimate the number of ;nodes
needed in the clusters based on historical data.

;=====

to computeExpectedValues

  set expectedValues array:from-list n-values numberOfClusterTypes [0]

  let i 0

  let temp 0

  while [i < numberOfClusterTypes][

    set overallNumberOfNeededClusters (array:item numberOfNeededClusters i +
overallNumberOfNeededClusters )

    set i (i + 1)

  ]

  set i (0)

```

```

while [i < numberOfClusterTypes][
    set temp precision ((array:item numberOfNeededClusters
i)/(overallNumberOfNeededClusters )) 3;; expected value of each RoA

    array:set expectedValues i temp

    set i (i + 1)

]

;print expectedValues

end

;=====

; This method computes the number of clusterheads based on the input.

;=====

to findNumberOfClusterHeads

    let temp 1

    let i 0

    let ch 0

while [i < numberOfClusterTypes ][

    set temp (precision ((array:item expectedValues i) * (array:item RoA i) *
normalizationFactor) 0) + temp

    set i i + 1

]

set numberOfClusterHeads precision (400 / temp) 0

end

;=====

; This assigns the respective RoA to each cluster.

;=====

to assignDistancesToClusterHeads

    let clusterNumber 0

    let temp 0

    let i 0

```

```

let j 0
while [ i < numberOfClusterTypes] [
  set temp (precision ((array:item expectedValues i) * numberOfclusterHeads) 0)
  repeat temp [
    ; print [member] of array:item cluster_heads j

    ask array:item cluster_heads j [set rangeOfAdvertise (array:item RoA i) *
granularityFactor + (random granularityFactor -
      (granularityFactor / 2) + 0.5) set member 1 set CHMembers array:from-list n-
values (30) ["null"]]

    set j (j + 1)

    ; ask array:item cluster_heads i [print (array:item RoA i) * granularityFactor
;]

  ]

  set i ( i + 1)

]

clusterInitial
end

;=====
;This procedure initializes the clusterheads.
;=====

to clusterInitial
  set p 0 set x 1
  set level array:from-list n-values numberOfClusterHeads[0]
  set clusterHeadsBlock table:make
  while [p < numberOfClusterHeads][
    table:put clusterHeadsBlock p array:from-list n-values 9[false]
    set p p + 1
  ]
;print clusterHeadsBlock

set p 0

set startingCluster 0 ; The variable starting cluster is used to simulate the
;parallel advertisement

```

```

    joinToClusters3
end

;=====

; This is the last version of the procedure that is used to join the nodes to the
;clusters.

;=====

to joinToClusters3

    let temp 0
    let tempBlock 0
    let nextAdvertise array:from-list n-values numberOfClusterHeads["right"]
    let currentAdvertise "right"
    let turtleNumber 0
    let clusterNumber startingCluster
    let clusterNumberTemp 0
    let allClustersZero 0
    let level' 0
    let notfinished true

while [clusterNumberTemp < numberOfClusterHeads][

    set level' array:item level clusterNumber ; The layers of advertisement
    if 0 < ([number] of array:item cluster_heads clusterNumber + level')
    and([number] of array:item cluster_heads clusterNumber + level') < 400 [
        if [joined] of turtle ([number] of array:item cluster_heads clusterNumber +
level') = clusterNumber
            or [beingCH] of turtle ([number] of array:item cluster_heads clusterNumber +
level') = 1
                [
                    set currentMainTurtle turtle ([number] of array:item cluster_heads clusterNumber +
level')

                    set turtleNumber ([number] of currentMainTurtle)

                    if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member] of
array:item cluster_heads clusterNumber [

                        set currentAdvertise (array:item nextAdvertise clusterNumber )

```

```

;;right
if currentAdvertise = "right" ;Advertisement to the node on the right

ifelse [edge] of currentMainTurtle != "right" ; Check for the boundary
  and [edge] of currentMainTurtle != "right-up"
  and [edge] of currentMainTurtle != "right-down"
  [

    set temp ([number] of currentMainTurtle + 20 )

    if [joined] of turtle temp = "null"
      and [beingCH] of turtle temp = "null"[
        ;;print ["right " ]

        ask turtle temp [set joined clusterNumber set color clusterNumber * 10 +
5]

        ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]
      ]

    ][

    set tempBlock table:get clusterHeadsBlock clusterNumber

    array:set tempBlock 1 true
    array:set tempBlock 5 true
    array:set tempBlock 8 true

    table:put clusterHeadsBlock clusterNumber tempBlock

    ]

    set currentAdvertise ("up")

  ]

```

```

]

if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member] of
array:item cluster_heads clusterNumber[

if currentAdvertise = "up"           ;Advertisement to the node on the right

ifelse [edge] of currentMainTurtle != "up" ; check for the boundary

and [edge] of currentMainTurtle != "right-up"

and [edge] of currentMainTurtle != "left-up"

[

set temp ( [number] of currentMainTurtle + 1)

if [joined] of turtle temp = "null"

and [beingCH] of turtle temp = "null"[

;; print "up"

ask turtle temp [set joined clusterNumber set color ((clusterNumber * 10
+ 5))]

ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]

; ask array:item cluster_heads clusterNumber [set rangeOfAdvertise
[rangeOfAdvertise] of array:item cluster_heads clusterNumber - 1]

]

][

set tempBlock table:get clusterHeadsBlock clusterNumber

array:set tempBlock 2 true

array:set tempBlock 5 true

array:set tempBlock 6 true

table:put clusterHeadsBlock clusterNumber tempBlock

]

]

set currentAdvertise ("left")

]

```

```

    if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member] of
array:item cluster_heads clusterNumber[
    if currentAdvertise = "left" [;Advertisement to the node on the left

    ifelse [edge] of currentMainTurtle != "left" ; Check for the boundary
    and [edge] of currentMainTurtle != "left-up"
    and [edge] of currentMainTurtle != "left-down"
    [
    set temp ( [number] of currentMainTurtle - 20)
    if [joined] of turtle temp = "null"
        and [beingCH] of turtle temp = "null"[
        ;;print "left"
        ask turtle temp [set joined clusterNumber set color clusterNumber * 10 +
5]

        ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]

        ; ask array:item cluster_heads clusterNumber [set rangeOfAdvertise
[rangeOfAdvertise] of array:item cluster_heads clusterNumber - 1]
        ]
    ]]
    set tempBlock table:get clusterHeadsBlock clusterNumber
    array:set tempBlock 3 true
    array:set tempBlock 6 true
    array:set tempBlock 7 true
    table:put clusterHeadsBlock clusterNumber tempBlock
    ]
]
set currentAdvertise ("down")
]
if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member] of
array:item cluster_heads clusterNumber[
    let m 0

```



```

if currentAdvertise = "down" [;Advertisement to the node below

    ifelse [edge] of currentMainTurtle != "down" ; check for the boundary
    and [edge] of currentMainTurtle != "left-down"
    and [edge] of currentMainTurtle != "right-down"
    [
        set temp ( [number] of currentMainTurtle - 1)
        if [joined] of turtle temp = "null"
            and [beingCH] of turtle temp = "null"[
                ;;print "down"
                ask turtle temp [set joined clusterNumber set color clusterNumber * 10 +
5]

                ask array:item cluster_heads clusterNumber [set member member + 1
array:set CHMembers member - 1 temp]

                ; ask array:item cluster_heads clusterNumber [set rangeOfAdvertise
[rangeOfAdvertise] of array:item cluster_heads clusterNumber - 1]

            ]

        ]

    ]

    set tempBlock table:get clusterHeadsBlock clusterNumber

    array:set tempBlock 4 true
    array:set tempBlock 7 true
    array:set tempBlock 8 true

    table:put clusterHeadsBlock clusterNumber tempBlock

    ]

    ]

    set currentAdvertise ("right")

    ]

    ]

    ]

    set clusterNumber (clusterNumber + 1)

```

```

        set clusterNumbertemp (clusterNumbertemp + 1)
    if clusterNumber >= numberOfClusterHeads [
        set clusterNumber (clusterNumber - numberOfClusterHeads)

    ]
]

;*****

; This is the recursive part of the joiningCluster3 procedure that advertises the
;needed number of nodes after the first round of advertisement.

;*****

set clusterNumber startingCluster

set clusterNumbertemp 0

set p p + 1

while [clusterNumberTemp < numberOfClusterHeads][

    set currentMainTurtle turtle ([number] of array:item cluster_heads
clusterNumber)

    set tempBlock table:get clusterHeadsBlock clusterNumber

    if p = 1 and array:item tempBlock 1 = false[

set temp [number] of currentMainTurtle + 20 * (x - 1)

ifelse [edge] of turtle temp != "right"

and [edge] of turtle temp != "right-up"

and [edge] of turtle temp != "right-down"

[

array:set level clusterNumber 20 * x

][

array:set tempBlock 1 true

]

]
]

```

```

]
if p = 2 and array:item tempBlock 2 = false[
set temp ([number] of currentMainTurtle + 1 * (x - 1))
ifelse [edge] of turtle temp != "up"
and [edge] of turtle temp != "right-up"
and [edge] of turtle temp != "left-up"
  [array:set level clusterNumber 1 * x ][
  array:set tempBlock 2 true
]
]

if p = 3 and array:item tempBlock 3 = false[
set temp ([number] of currentMainTurtle + -20 * (x - 1))
ifelse [edge] of turtle temp != "left"
and [edge] of turtle temp != "left-up"
and [edge] of turtle temp != "left-down"
  [ array:set level clusterNumber -20 * x ]
  [array:set tempBlock 3 true
]
]

if p = 4 and array:item tempBlock 4 = false[
ifelse [edge] of turtle temp != "down"
and [edge] of turtle temp != "left-down"
and [edge] of turtle temp != "right-down"
  [array:set level clusterNumber -1 * x ][
  array:set tempBlock 4 true
]
]

```

```

]
]
if p = 5 and array:item tempBlock 5 = false[
set temp ([number] of currentMainTurtle + 20 * (x - 1) + 1)
ifelse [edge] of turtle temp != "right"
and [edge] of turtle temp != "right-up"
and [edge] of turtle temp != "right-down"
and [edge] of turtle temp != "left-up"
and [edge] of turtle temp != "up"
[
array:set level clusterNumber 20 * x + 1
][
array:set tempBlock 5 true
]
]
if p = 6 and array:item tempBlock 6 = false[
set temp ([number] of currentMainTurtle + -20 * (x - 1) + 1)
; print temp
ifelse [edge] of turtle temp != "left"
and [edge] of turtle temp != "left-up"
and [edge] of turtle temp != "left-down"
and [edge] of turtle temp != "up"
and [edge] of turtle temp != "right-up"
[
array:set level clusterNumber -20 * x + 1
][
array:set tempBlock 6 true
]
]
if p = 7 and array:item tempBlock 7 = false[

```

```

set temp ([number] of currentMainTurtle + -20 * (x - 1) - 1)
if temp > -1[

  ifelse [edge] of turtle temp != "left"
  and [edge] of turtle temp != "left-down"
  and [edge] of turtle temp != "right-down"
  and [edge] of turtle temp != "left-up"
  and [edge] of turtle temp != "down"

  [
    array:set level clusterNumber -20 * x - 1
  ]

  [
    array:set tempBlock 7 true
  ]

  ]

]

if p = 8 and array:item tempBlock 8 = false[

  set temp ([number] of currentMainTurtle + 20 * (x - 1) - 1)
  if temp > -1[
    ifelse [edge] of turtle temp != "right"
    and [edge] of turtle temp != "left-down"
    and [edge] of turtle temp != "right-down"
    and [edge] of turtle temp != "right-up"
    and [edge] of turtle temp != "down"

    [
      array:set level clusterNumber 20 * x - 1
    ]
    [array:set tempBlock 8 true

  ]

  ]
]

```

```

]
    table:put clusterHeadsBlock clusterNumber tempBlock
    set clusterNumber (clusterNumber + 1)
    set clusterNumbertemp (clusterNumbertemp + 1)
if clusterNumber >= numberOfClusterHeads [
    set clusterNumber clusterNumber - numberOfClusterHeads
]
]
,*****
    set clusterNumber 0

    set notFinished false

    while [clusterNumber < numberOfClusterHeads][
        ;print [rangeOfAdvertise] of array:item cluster_heads clusterNumber

        if [rangeOfAdvertise] of array:item cluster_heads clusterNumber > [member]
of array:item cluster_heads clusterNumber [
            set notFinished true
        ]

        set clusterNumber clusterNumber + 1
    ]

    ifelse p < 9
    [
        ifelse startingCluster < numberOfClusterHeads - 1[
            set startingCluster startingCluster + 1
        ][
            set startingCluster 0
        ]
        joinToClusters3]
    [
        ifelse notFinished = true and x < 5[

```

```
    set p 0
    set x x + 1
    ifelse startingCluster < numberOfClusterHeads[
set startingCluster startingCluster + 1
][
set startingCluster 0
]
    joinToClusters3
]
[
;isolatedNodes ; should be uncommented for extension 2, method 1
reclustering1 ; re-selecting clusterheads
]
]
end
```

```

;=====
; This is one version of the reclustering procedures. In this procedures, after one
clustering, the center of the clusters are found and assigned as new clusterheads. The
clustering algorithm is executed again for the new clusterheads.
;=====

to reclustering1

ifelse totalDistances > 0 and (totalDistances < 500 and iterations < 200) [
    ;isolatedNodes

    let n 0
    let m 0
    ; These matrices are used to calculate the central nodes
    let maxRow-turtle array:from-list n-values numberOfClusterHeads[0]
    let minRow-turtle array:from-list n-values numberOfClusterHeads[20]
    let maxColumn-turtle array:from-list n-values numberOfClusterHeads[0]
    let minColumn-turtle array:from-list n-values numberOfClusterHeads[19]
    let tempNum 0
    while [n < 400] [

        if [joined] of turtle n != "null" [
            set tempNum precision (([number] of turtle n) / 20) 0
            if tempNum > ([number] of turtle n) / 20
            [
                set tempNum tempNum - 1
            ]

            if tempNum > (array:item maxColumn-turtle [joined] of turtle n) [
                array:set maxColumn-turtle ([joined] of turtle n) tempNum
            ]
        ]
    ]
]

```



```

]
if tempNum < (array:item minColumn-turtle [joined] of turtle n)[
  array:set minColumn-turtle [joined] of turtle n tempNum
]

if (([number] of turtle n) - tempNum * 20) > (array:item maxRow-turtle
[joined] of turtle n) [
  array:set maxRow-turtle [joined] of turtle n ([number] of turtle n -
tempNum * 20)
]

if (([number] of turtle n) - tempNum * 20) < (array:item minRow-turtle
[joined] of turtle n) [
  array:set minRow-turtle [joined] of turtle n ([number] of turtle n -
tempNum * 20)
]
]

ask turtle n [set member "null" set joined "null" set color white set
size 1.5 set beingCH "null"]

set n n + 1
]

set level array:from-list n-values numberOfClusterHeads[0]
set p 0
while [p < numberOfClusterHeads ][
  table:put clusterHeadsBlock p array:from-list n-values 9[false]
  set p p + 1
]

set x 1 set p 0
set m 0
let tempClusterHeads array:from-list n-values numberOfClusterHeads[0]
while [m < numberOfClusterHeads][
  array:set tempClusterHeads m array:item cluster_heads m

```

```

        set m m + 1
    ]

    set m 0

    while [m < numberOfClusterHeads][

        set tempNum (precision ((array:item maxColumn-turtle m + array:item
minColumn-turtle m) / 2) 0)

        ;print tempNum

        if tempNum > (array:item maxColumn-turtle m + array:item minColumn-
turtle m) / 2

        [

            set tempNum tempNum - 1

        ]

        let tempNum1 tempNum * 20 + precision ((array:item maxRow-turtle m +
array:item minRow-turtle m) / 2) 0

        if tempNum1 > tempNum * 20 + (array:item maxRow-turtle m + array:item
minRow-turtle m) / 2

        [

            set tempNum1 tempNum1 - 1

        ]

        ; print tempNum1

        ;print minColumn-turtle

        ifelse [beingCH] of turtle tempNum1 != 1[

            array:set cluster_heads m turtle tempNum1

        ]

        ifelse tempNum < 399 [

```

```

array:set cluster_heads m turtle (tempNum1 + 1)
][array:set cluster_heads m turtle (tempNum1 - 1)
]
]

ask array:item cluster_heads m [set member 1 set beingCH 1 set Color
green set size 2]

set m m + 1
]
set m 0
set iterations iterations + 1
while [m < numberOfClusterHeads][
  if [rangeOfAdvertise] of array:item cluster_Heads m > 5
    [ask array:item cluster_Heads m [set beingCH "null"]
      let tempTurtle one-of turtles
      while [[beingCH] of tempTurtle != "null" ][
        set tempTurtle one-of turtles
      ]
      array:set cluster_heads m tempTurtle
      ask array:item cluster_heads m [ set beingCH 1 set Color green
set size 2 set member 1]
    ]
    set m m + 1
]
set-current-plot "CHChanges"
let i 0
let tempX 0
let tempY 0
while [i < numberOfClusterHeads][
  set tempX [xcor] of array:item cluster_Heads i
  set tempY [ycor] of array:item cluster_Heads i

```

```

    ifelse tempX > [xcor] of array:item tempClusterHeads i [
      set tempX tempX - [xcor] of array:item tempClusterHeads i
    ]
    set tempX [xcor] of array:item tempClusterHeads i - tempX
  ]
  ifelse tempY > [ycor] of array:item tempClusterHeads i [
    set tempY tempY - [ycor] of array:item tempClusterHeads i
  ]
  set tempY [ycor] of array:item tempClusterHeads i - tempY
  ]
  set totalDistances totalDistances + tempX + tempY
  set i i + 1
]
plot totalDistances
assignDistancesToClusterHeads

][if totalDistances > 500 or iterations > 200[

  start
  ]
]

end

;=====
; This is another version of the reclustering method.
;=====

to reclustering

ifelse totalDistances > 0 and (totalDistances < 500 and iterations < 200)[
;Checks for the threshold values

  let n 0

```

```

let m 0

; These matrices are used to calculate the central nodes

let maxRow-turtle array:from-list n-values numberOfClusterHeads[0]
let minRow-turtle array:from-list n-values numberOfClusterHeads[20]
let maxColumn-turtle array:from-list n-values numberOfClusterHeads[0]
let minColumn-turtle array:from-list n-values numberOfClusterHeads[19]
set central-turtle array:from-list n-values numberOfClusterHeads[1000]
let centerXCors array:from-list n-values numberOfClusterHeads[0]
let centerYCors array:from-list n-values numberOfClusterHeads[0]
let tempNum 0

; centralNodes

set n 0

while [n < 400] [

if [joined] of turtle n != "null" [

set tempNum precision (([number] of turtle n) / 20) 0

if tempNum > ([number] of turtle n) / 20

[

set tempNum tempNum - 1

]

if tempNum > (array:item maxColumn-turtle [joined] of turtle n) [

array:set maxColumn-turtle ([joined] of turtle n) tempNum

]

if tempNum < (array:item minColumn-turtle [joined] of turtle n) [

array:set minColumn-turtle [joined] of turtle n tempNum

]

if (([number] of turtle n) - tempNum * 20) > (array:item maxRow-turtle
[joined] of turtle n) [

```

```

        array:set maxRow-turtle [joined] of turtle n ([number] of turtle n -
tempNum * 20)
    ]

    if (([number] of turtle n) - tempNum * 20) < (array:item minRow-turtle
[joined] of turtle n) [

        array:set minRow-turtle [joined] of turtle n ([number] of turtle n -
tempNum * 20)
    ]
]

ask turtle n [set joined "null" set color white set size 1.5 set beingCH
"null" set member "null" set CHMembers "null"]

set n n + 1
]

set level array:from-list n-values numberOfClusterHeads[0]

set p 0
while [p < numberOfClusterHeads ][
    table:put clusterHeadsBlock p array:from-list n-values 9[false]
    set p p + 1
]

set x 1 set p 0

set m 0

let tempClusterHeads array:from-list n-values numberOfClusterHeads[0]
while [m < numberOfClusterHeads][
    array:set tempClusterHeads m array:item cluster_heads m
    set m m + 1
]

set m 0

while [m < numberOfClusterHeads][

```

```

        set tempNum (precision ((array:item maxColumn-turtle m + array:item
minColumn-turtle m) / 2) 0)

        ;print tempNum

        if tempNum > (array:item maxColumn-turtle m + array:item minColumn-
turtle m) / 2

            [

                set tempNum tempNum - 1

            ]

        let tempNum1 tempNum * 20 + precision ((array:item maxRow-turtle m +
array:item minRow-turtle m) / 2) 0

        if tempNum1 > tempNum * 20 + (array:item maxRow-turtle m + array:item
minRow-turtle m) / 2

            [

                set tempNum1 tempNum1 - 1

            ]

        ; print tempNum1

        ;print minColumn-turtle

        ifelse [beingCH] of turtle tempNum1 != 1[

array:set cluster_heads m turtle tempNum1

        ][

        ifelse tempNum < 399 [

array:set cluster_heads m turtle (tempNum1 + 1)

        ][

array:set cluster_heads m turtle (tempNum1 - 1)

                ]

        ]

        ;set cluster_heads central-turtle

```

```

ask array:item cluster_heads m [ set beingCH 1 set Color m * 10 + 5 set
size 2 set member 1 set rangeOfAdvertise [rangeOfAdvertise] of array:item
tempClusterHeads m

set CHMembers array:from-list n-values (30) ["null"]]

set m m + 1

]

set m 0

set iterations iterations + 1

while [m < numberOfClusterHeads][

if [rangeOfAdvertise] of array:item cluster_Heads m - [member] of
array:item cluster_heads m > 3

[ask array:item cluster_Heads m [set beingCH "null"]]

let tempTurtle one-of turtles

while [[beingCH] of tempTurtle != "null"][

set tempTurtle one-of turtles

]

array:set cluster_heads m tempTurtle

ask array:item cluster_heads m [ set beingCH 1 set Color m * 10
+ 5 set size 2 set member 1 set rangeOfAdvertise [rangeOfAdvertise] of array:item
tempClusterHeads m

set CHMembers array:from-list n-values (30) ["null"]]

]

set m m + 1

]

set-current-plot "CHChanges"

let i 0

let tempX 0

let tempY 0

while [i < numberOfClusterHeads][

set tempX [xcor] of array:item cluster_Heads i

```



```

set tempY [ycor] of array:item cluster_Heads i

ifelse tempX > [xcor] of array:item tempClusterHeads i [
  set tempX tempX - [xcor] of array:item tempClusterHeads i
][
  set tempX [xcor] of array:item tempClusterHeads i - tempX
]
ifelse tempY > [ycor] of array:item tempClusterHeads i [
  set tempY tempY - [ycor] of array:item tempClusterHeads i
][
  set tempY [ycor] of array:item tempClusterHeads i - tempX
]
set totalDistances totalDistances + tempX + tempY
set i i + 1
]
plot totalDistances
;print totalDistances
  ;print "hello"
let z 0
while [z < 400] [

  if [beingCH] of turtle z != 1[
    ask turtle z [set size 1.5]
  ]
  set z z + 1
]
if endddd != true[
clusterInitial
]
][if totalDistances > 500 or iterations > 200[

```

```

        if enddd != true[
            start
        ]
    ]

]

end

;=====
; This procedure takes care of the nodes that have not already joined any cluster.
;=====

to isolatedNodes

    set isolatedNode 0

    let clusterNumber 0

    let clusterToJoin 0

    let rowDistance 0

    let columnDistance 0

    let temp 0

    let distancee 400

    set nearestCluster 0

    let g 0

    let counter 0

    let firstNearest 0

    while [ isolatedNode < 400][
        while [g < numberOfClusterHeads][
            ask array:item cluster_heads g [set flag 0]

            set g g + 1

        ]

        if [joined] of turtle isolatedNode = "null" and [beingCH] of turtle isolatedNode
        = "null"[

```

```

        set nearestTemp isolatedNode

        findeNearestClusterHead

        set firstNearest nearestCluster

        ask array:item cluster_heads nearestCluster [set flag 1]

set counter 0

set endd false

        ifelse [rangeOfAdvertise] of array:item cluster_Heads nearestCluster -
[member] of array:item cluster_Heads nearestCluster > -3[

        ask turtle isolatedNode [ set joined nearestCluster set color nearestCluster
* 10 + 5]

        ask array:item cluster_heads nearestCluster [set member member + 1 array:set
CHMembers member - 1 isolatedNode]

        set endd true

    ][

;,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

        while [[rangeOfAdvertise] of array:item cluster_Heads nearestCluster -
[member] of array:item cluster_Heads nearestCluster <= -3 and endd != true][

            ; type [rangeOfAdvertise] of array:item cluster_Heads nearestCluster type
"      ," type [member] of array:item cluster_Heads nearestCluster print ""

            set counter counter + 1

            set nearestTemp nearestCluster

            set nearestCluster 0

            findeNearestClusterHead

            print [rangeOfAdvertise] of array:item cluster_Heads nearestCluster -
[member] of array:item cluster_Heads nearestCluster

            ask array:item cluster_heads nearestTemp [set flag 1]

            let i 0

            let nearestNode 0

            while [[CHMembers] of array:item cluster_heads nearestTemp != "null" and i <
[member] of array:item cluster_heads nearestTemp][

```

```

    set columnDistance [ycor] of array:item cluster_heads nearestCluster
    set temp [ycor] of turtle i
    set columnDistance abs (columnDistance - temp)
    set temp [xcor] of array:item cluster_heads clusterNumber
    set rowDistance [ycor] of turtle nearestTemp
    set rowDistance abs(rowDistance - temp )
    set temp rowDistance + columnDistance
    if temp < distancee [
        set distancee temp
        set nearestNode i
    ]

    set i i + 1
]

ask turtle nearestNode [set joined nearestCluster set color nearestCluster *
10 + 5]

ask array:item cluster_heads nearestTemp [set member member - 1 array:set
CHMembers member - 1 "null"]

ask array:item cluster_heads nearestCluster [set member member + 1 array:set
CHMembers member - 1 nearestNode]

    set distancee 400
]

]

]

set distancee 400

set temp 0

set nearestCluster 0

set clusterNumber 0

```

```

        findeNearestClusterHead
        set isolatedNode isolatedNode + 1
    ]
;*****
end

;=====
; This procedure is used to output the results.
;=====

to printNumberOfMembers
let m 0

set NOMembers array:from-list n-values 6 [0]
    set m 0
while [m < numberOfClusterHeads ][
ifelse [member] of array:item cluster_heads m < 5 [
    array:set NOMembers 1 array:item NOMembers 1 + 1
][ifelse [member] of array:item cluster_heads m < 10 [
    array:set NOMembers 2 array:item NOMembers 2 + 1
][ifelse [member] of array:item cluster_heads m < 15 [
    array:set NOMembers 3 array:item NOMembers 3 + 1
][
    ifelse [member] of array:item cluster_heads m < 20 [
        array:set NOMembers 4 array:item NOMembers 4 + 1
    ][
        array:set NOMembers 5 array:item NOMembers 5 + 1
    ]
]
]
]
]
]

```

```
set m m + 1
]
test
end
;=====
; This procedure checks to see if the clustering algorithm is working correctly. In
;other words, it checks is the number of nodes taken by each cluster against the
;number of needed nodes in that cluster.
;=====
to test
set NOMembers array:from-list n-values 13 [0]
let m 0
while [m < numberOfClusterHeads ][
ifelse [member] of array:item cluster_heads m < 7 [
array:set NOMembers 1 array:item NOMembers 1 + 1
][ifelse [member] of array:item cluster_heads m < 10 [
array:set NOMembers 2 array:item NOMembers 2 + 1

][ifelse [member] of array:item cluster_heads m < 13 [
array:set NOMembers 3 array:item NOMembers 3 + 1
][
ifelse [member] of array:item cluster_heads m < 16 [
array:set NOMembers 4 array:item NOMembers 4 + 1
][
ifelse [member] of array:item cluster_heads m < 19[
array:set NOMembers 5 array:item NOMembers 5 + 1
]
[
ifelse [member] of array:item cluster_heads m < 22 [
array:set NOMembers 6 array:item NOMembers 6 + 1
]
]
```



```

; array:set AVGNOMembers m array:item NOMembers m

set m m + 1
]
]
set m 1
repeat 10[

array:set AVGNOMembers m ((array:item NOMembers m + array:item AVGNOMembers m))

set m m + 1
]
set m 0
repeat numberOfClusterHeads [
type [member] of array:item cluster_heads m type ", "

set m m + 1
]

print " "
set m 0
repeat numberOfClusterHeads [
type [rangeOfAdvertise] of array:item cluster_heads m type ", "

set m m + 1
]
print " "
print "=====; "
Let shapeArr array:from-list n-values numberOfClusterHeads [0]
array:set shapeArr 0 "star"
array:set shapeArr 1 "circle"
array:set shapeArr 2 "triangle"

```



```

array:set shapeArr 3 "pentagon"
array:set shapeArr 4 "x"
array:set shapeArr 5 "leaf"
array:set shapeArr 6 "plant"

set m 0

repeat 400 [

if [joined] of turtle m != "null"

[ask turtle m [set shape array:item shapeArr (joined mod 6) set size 0.7 set color
([joined] of turtle m) * 10 + 5]]

set m m + 1

]

set m 0

repeat numberOfClusterHeads [

ask array:item cluster_heads m [set shape array:item shapeArr (m mod 6) set size 1
set color m * 10 + 5]

set m m + 1

]

end

;=====

; This is a procedure that is used in the crawling method. Two different methods are
;tested for finding the nearest clusterhead.

;=====

to findeNearestClusterHead

  let clusterNumber 0

  let distancee 400

  let rowDistance 0

  let columnDistance 0

  let temp 0

```

```

set nearestCluster 0

while[clusterNumber < numberOfClusterHeads][

    ; type nearestCluster print " "
; type clusterNumber type "      " print nearestTemp
;test
if clusterNumber != nearestTemp[

set columnDistance [ycor] of array:item cluster_heads clusterNumber
set temp [ycor] of turtle nearestTemp

set columnDistance abs (columnDistance - temp)

    set temp [xcor] of array:item cluster_heads clusterNumber
set rowDistance [xcor] of turtle nearestTemp

    set rowDistance abs(rowDistance - temp )

set temp rowDistance + columnDistance

    ifelse temp < distancee and ([flag] of array:item cluster_heads
clusterNumber) = 0[

        if check = true[

]

set distancee temp

set nearestCluster clusterNumber

type nearestCluster type " , "

][if temp = distancee[

    if ([rangeOfAdvertise] of (array:item cluster_Heads nearestCluster) -
([member] of array:item cluster_heads nearestCluster)) <

```

```

        ([rangeOfAdvertise] of array:item cluster_Heads clusterNumber -
[member] of array:item cluster_heads clusterNumber) and

            [flag] of array:item cluster_heads clusterNumber = 0 [
                set nearestCluster clusterNumber
            ]
        ]
    ]
]
set clusterNumber ClusterNumber + 1
]
if nearestCluster = nearestTemp[
set endd true
]
set clusternumber 0
while[clusterNumber < numberOfClusterHeads][
; type [flag] of array:item cluster_heads clusterNumber type " , "
set clusterNumber ClusterNumber + 1
]
print " "
    print nearestCluster
end
;=====
; This is another version of attaching the isolated nodes to the clusters.
;=====
to unjoinedNodes1
let shortestPaths array:from-list n-values numberOfClusterHeads ["null"]
let nodee 0
let fin false
while [fin = false][
set fin true

```

```

while [nodee < 400][

if ([joined] of turtle nodee = "null")[

;::::::::::::::::::::::::::::::::::::finding the neediest clusterhead

set fin false

let clusterNumber 0

let neediest 0

while [clusterNumber < numberOfClusterHeads][

ask array:item cluster_heads clusterNumber [set joined clusterNumber ]

if ( [rangeOfadvertise] of array:item cluster_heads clusterNumber - [member] of
array:item cluster_heads clusterNumber

> [rangeOfadvertise] of array:item cluster_heads neediest - [member] of
array:item cluster_heads neediest )[

set neediest clusterNumber

]

set clusterNumber clusterNumber + 1

]

;::::::::::::::::::::::::::::::::::::find the nearest cluster to the node

let tempNode nodee

findeNearestClusterHead1

;::::::::::::::::::::::::::::::::::::find the shortest path from nearest clusterhead to
the neediest cluster (greedy approach)

set firstNode nodee

set lastNode [number] of array:item cluster_heads neediest

shortestpath

]

set nodee nodee + 1

```

```

]
]
end

;=====
; This is a procedure that is used in the crawling method.
;=====

to findeNearestClusterHead1

  let clusterNumber 0
  let distancee 400
  let rowDistance 0
  let columnDistance 0
  let tempNode 0
  let temp 0
  set nearestCluster 0
  while[clusterNumber < numberOfClusterHeads][
    if clusterNumber != tempNode[
      set columnDistance [ycor] of array:item cluster_heads clusterNumber
      set temp [ycor] of turtle nearestTemp
      set columnDistance abs (columnDistance - temp)
      set temp [xcor] of array:item cluster_heads clusterNumber
      set rowDistance [ycor] of turtle nearestTemp
      set rowDistance abs(rowdistance - temp )
      set temp rowDistance + columnDistance
      if temp < distancee and ([flag] of array:item cluster_heads
clusterNumber) = 0[
        set distancee temp
        set nearestCluster clusterNumber
      ]
    ]
  ]
]

```

```

    set clusterNumber ClusterNumber + 1
  ]
  set clusterNumber 0
end

;=====
;This procedure finds the shortest path between the node that is crawling and the
cluster ;that needs the node.
;=====

to shortestPath
  let Hdirection 0
  let Vdirection 0

  let rowDistance abs ( [ycor] of turtle firstNode - [ycor] of turtle lastNode)
  let columnDistance abs ( [xcor] of turtle firstNode - [xcor] of turtle lastNode)
  let i 0
  let j 0
  let currentNode firstNode
  let nextNode 0
  ;ask turtle firstNode [set color red]
  ;ask turtle lastNode [set color red]
  ifelse [xcor] of turtle firstNode > [xcor] of turtle lastNode[
    set Hdirection -1
  ][
    set Hdirection 1
  ]
  ifelse [ycor] of turtle firstNode > [ycor] of turtle lastNode[
    set Vdirection -1
  ][
    set Vdirection 1
  ]
]

```

```

set i 0

;print lastNode

while [i < columnDistance and [joined] of turtle currentNode != [joined] of turtle
lastnode][

    set nextNode currentNode + hdirection * 20

    if [joined] of turtle nextNode != [joined] of turtle currentNode[

        ask turtle currentNode [set joined [joined] of turtle nextNode set color [color]
of turtle nextNode]

    ]

    set currentnode nextNode

set i i + 1

]

set i 0

while [i < rowDistance and [joined] of turtle currentNode != [joined] of turtle
lastnode][

    set nextNode currentNode + vdirection

        ask turtle currentNode [set joined [joined] of turtle nextNode set color
[color] of turtle nextNode]

        set currentnode nextNode

set i i + 1

]

ask turtle lastNode [set member member + 1 ]end

```

VITA

Alireza Boloorchi Tabrizi

Candidate for the Degree of

Master of Science

Thesis: AN ADAPTIVE CLUSTERING ALGORITHM FOR WIRELESS SENSOR  
NETWORKS

Major Field: Computer Science

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December 2011.

Completed the requirements for the Bachelor of Science in Computer Engineering at Amirkabir University of Technology, Tehran, Iran in December 2008.