P - 1636

CLUSTERING IN WIRELESS SENSOR NETWORKS

A PROJECT REPORT

Submitted by

JITEN.R	**71202205019**
SURESH.K	**71202205050**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2006

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**CLUSTERING IN WIRELESS SENSOR NETWORKS**" is the bonafide work of

"Mr. JITEN.R - 71202205019 and

Mr. SURESH.K - 71202205050"

who carried out the project work under my supervision.

[illegible]

SIGNATURE

Dr.G.Gopalsamy

HEAD OF THE DEPARTMENT

Information Technology,
Kumaraguru College of Technology,
Coimbatore – 641006.

[illegible] 24/4/06

SIGNATURE

Ms.L.S.Jayashree

SUPERVISOR

Assistant Professor

Computer Science and Engineering,
Kumaraguru College of Technology,
Coimbatore – 641006.

The candidates with the University Register Nos. **71202205019, 71202205050 were** examined by us in the project viva-voce examination held on.....26.04.2006............

[illegible]

INTERNAL EXAMINER

[illegible] 26/4/06

EXTERNAL EXAMINER

DECLARATION

We,

JITEN.R	**71202205019**
SURESH.K	**71202205050**

Declare that the project entitled **"CLUSTERING IN WIRELESS SENSOR NETWORKS"**, submitted in partial fulfillment to Anna University as the project work of Bachelor Of Technology (Information Technology) Degree, is a record of original work done by us under the supervision and the guidance of **Ms.L.S. Jayashree, M.E., Assistant Professor**, Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore.

Place: Coimbatore

Date: 24 4.06

Jiten.R 24/4/06

[Jiten.R]

[illegible] 24/4/06

[Suresh.K]

Project Guided by

[illegible]

[Ms.L.S Jayashree, M.E.,]

ACKNOWLEDGEMENT

We express our sincere thanks to **Dr.K.Arumugam, B.E. (Hons), M.S. (USA), MIE.,** Correspondent, Kumaraguru College of Technology and the management, for providing us this opportunity to undertake this project work.

We express our profound gratitude to **Dr.K.K.Padmanabhan B.Sc. (Engg)., M.Tech., Ph.D.,** Principal, Kumaraguru College of Technology, Coimbatore, for permitting us to undergo a project work.

We are greatly indebted to **Dr. G.Gopalsamy, Ph.D.,** Professor and Head of the Department of Information Technology, for the immense support he has provided us throughout our project.

We extend our sincere thanks to our project coordinator **Prof.K.R.Baskaran, B.E., M.S.,** Assistant Professor, Department of Information Technology, for his constant support and encouragement.

We would like to express our heart felt thanks to our guide, Ms. **L.S. Jayashree, M.E.,** Assistant Professor, Department of Computer Science and Engineering, who gave the initial idea and guidance to carry out every further step and for her everlasting counseling and untiring help throughout our project.

We sincerely thank our class advisor, **Ms.N.Suganthi, M.E.,** Senior Lecturer, Department of Information Technology, for her unconditional support and motivation in helping us in completing the project.

We would also like to thank all the faculty members and lab technicians of Department of Information Technology for their valuable guidance, support and encouragement during the course of our project work.

We like to express our deep sense of gratitude to our families and friends whose help aided the successful completion of the project.

ABSTRACT

This project titled "CLUSTERING IN WIRELESS SENSOR NETWORKS" looks into the details pertaining to building a protocol which will overcome the disadvantages of the existing clustering protocols.

In the case of physical clustering, we implemented the HEED (Hybrid Energy Efficient Distributed) protocol, in which cluster heads are selected periodically according to a hybrid of their residual energy and a secondary parameter, such as node proximity to its neighbors or node degree. HEED operates in quasi-stationary networks where nodes are location-unaware and have equal significance.

In the case of spatial clustering, the network is partitioned into a set of spatial regions with similar observations. Data models are build locally at each node and cluster is formed based on model coefficients (rather than on raw data). These co-efficient are used to capture trends and seasonalties in spatial regions. Scientists can gain insight into the overall distribution patterns and interesting correlations among sensory attributes by grouping such spatial regions which have observed into a single class.

TABLE OF CONTENTS

CHAP.NO	TITLE	PAGE.NO
	ABSTRACT	v
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
1.	INTRODUCTION	1
	1.1 Existing System and Its Limitations	1
	1.2 Proposed System and Its Advantages	1
2.	SYSTEM REQUIREMENT ANALYSIS	3
	2.1 Project Plan	3
	2.2 Software Requirements Specification	3
	2.2.1 Purpose	3
	2.2.2 Scope	3
	2.2.3 Specific Requirements	4
	2.2.3.1 Hardware Requirements	4
	2.2.3.2 Software Requirements	4
3.	SYSTEM STUDY	5
	3.1 Introduction	5
	3.2 Clustering in wireless sensors networks	8
	3.2.1 Challenges in clustering	9
	3.2.2 Types of clustering	10
	3.3 HEED Protocol	12
	3.3.1 Clustering parameters	12
	3.3.2 Protocol operation	13
	3.3.3 Pseudo code	15
	3.3.4 Results	16

	3.4 Spatial Clustering	18
	3.4.1 Introduction	18
	3.4.2 Feature modeling and distance	20
	3.4.3 TAO Project	20
	3.4.4 Spanning tree clustering	24
	3.4.5 Pseudo code	26
	3.4.6 Results	27
4.	**DESIGN DOCUMENTS**	29
	4.1 Flow diagram	29
5.	TESTING	30
6	FUTURE ENHANCEMENTS	31
7	CONCLUSION	32
8	APPENDIX	33
	8.1 Sample Code	33
	8.2 Sample Output	52
9	REFERENCES	57

LIST OF TABELS

SERIAL NO.	TABEL NAME	TABEL No	PAGE No
1.	Lifetime of HEED protocol	3.3.4.1	16
2.	TAO data	3.4.3.1	22
3.	Sensor locations	3.4.3.2	23
4.	E-δ Analysis	3.4.6.1	27

LIST OF FIGURES

SERIAL NO.	FIGURE NAME	FIG. No	PAGE No
1.	Components of a sensor node	3.1.1	6
2.	Methods of clustering	3.2.1	9
3.	Node Lifetime	3.3.4.1	17
4.	Network Lifetime	3.3.4.2	17
5.	Epsilon deleted graph	3.4.1	19
6.	Spanning tree clustering	3.4.4.1	25
7.	Delta analysis	3.4.6.1	28
8.	Epsilon analysis	3.4.6.2	28

1. INTRODUCTION:

1.1 The Existing System and its limitations

When we consider physical clustering, existing protocols [3] proposed in the literature minimize energy consumption on routing paths. While these approaches increase energy efficiency, they do not necessarily prolong network lifetime if certain nodes are "popular," i.e., present on most forwarding paths in the network. Even if dynamic routing (in which data is forwarded to nodes with the highest residual energy) is used, it may cause such problems as unbounded delay and routing loops. With clustering, a popular node is guaranteed to "lose its popularity" (no longer serve as cluster head) after a fixed interval of time. Of course, node popularity due to interest in the data it provides can only be reduced by deploying several redundant nodes, and rotating among them.

When we consider the spatial clustering approach, the existing system forms clusters based on the raw data and not on the underlying trend. This increases communication overhead. Further this system decreases the network lifetime. Existing system does not support in-network clustering.

1.2 Proposed System and its advantages.

In the case of physical clustering, the protocol HEED (Hybrid Energy-Efficient Distributed clustering), has four primary goals:

(i) Prolonging network lifetime by distributing energy consumption,

(ii) Terminating the clustering process within a constant number of iterations/steps,

(iii) Minimizing control overhead (to be linear in the number of nodes), and

(iv) Producing well-distributed cluster heads and compact clusters.

(v) HEED does not make any assumptions about the distribution or density of nodes, or about node capabilities, e.g., location-awareness.

In the case of spatial clustering, we group the spatial regions that have similar phenomenon into a single class. We build models on each node and cluster based on the model coefficients rather than on raw data. We implemented an in-network clustering approach and distributed algorithm that not only generates high quality clusters but also decreases the communication cost. Scientists can gain insight into the overall distribution patterns and interesting correlations among sensory attributes by grouping such spatial regions observed similar phenomenon into a single class.

2. SYSTEM REQUIREMENT ANALYSIS:

2.1 Project Plan

The requirement phase deals with the sequence of activities in producing the Software Requirements Specification (SRS) document. It must be ensured that all the requirements of the software are elicited and analyzed. In other words the needs of the system are identified.

In the problem analysis phase, the current system is analyzed by study of existing materials, and the changes to be made in the proposed system are decided upon. A clear understanding of the needs of the system must be framed. Analysis leads to the actual specification.

Designing aims at how to satisfy the needs of the system. The different modules of the system and the interaction between these modules to produce the desired functionality are identified. During detailed design, the internal logic of each module and their algorithmic design is specified. The major and important decisions are made in this phase.

Coding is the process of translating the design into code. The code developed must be easy to understand. Well-written code can reduce testing and maintenance effort.

Testing is done to check if the requirements of the user are met. It also involves in checking the functionality of each module as per the design document.

2.2 Software Requirement Specification

2.2.1 Purpose

The purpose of this project is to study the clustering problem in various applications and formulate a protocol that effectively clusters the sensors based on specified parameters.

2.2.2 Scope

Clustering of the sensors will be done on pre-specified parameters and the results will be analyzed in a simulated environment.

2.3. Specific Requirements

2.3.1 Hardware Requirements

Our system needs a minimum requirement of the following:

256 MB RAM

20 GB Hard disk

2.2 GHz Processor

2.3.2 Software Requirements

The softwares used are:

Language	:	C
OS	:	Windows Platform
Analysis tool	:	Microsoft Excel

3. SYSTEM STUDY

3.1 Introduction

Recent advances in micro-electro-mechanical systems technology, wireless communications, and digital electronics have enabled the development of low-cost, low-power, multifunctional sensor nodes that are small in size and communicate in short distances. These tiny sensor nodes, which consist of sensing, data processing, and communicating components, leverage the idea of sensor networks based on collaborative effort of a large number of nodes. Sensor networks represent a significant improvement over traditional sensors, which are deployed in the following two ways:

- Sensors can be positioned far from the actual phenomenon, i.e., something known by sense perception. In this approach, large sensors that use some complex techniques to distinguish the targets from environmental noise are required.
- Several sensors that perform only sensing can be deployed. The positions of the sensors and communications topology are carefully engineered. They transmit time series of the sensed phenomenon to the central nodes where computations are performed and data are fused.

A sensor network is composed of a large number of sensor nodes, which are densely deployed either inside the phenomenon or very close to it. The position of sensor nodes need not be engineered or pre-determined. This allows random deployment in inaccessible terrains or disaster relief operations. On the other hand, this also means that sensor network protocols and algorithms must possess self-organizing capabilities. Another unique feature of sensor networks is the cooperative effort of sensor nodes. Sensor nodes are fitted with an on-board processor. Instead of sending the raw data to the nodes responsible for the fusion, sensor nodes use their processing abilities to locally carry out simple computations and transmit only the required and partially processed data.

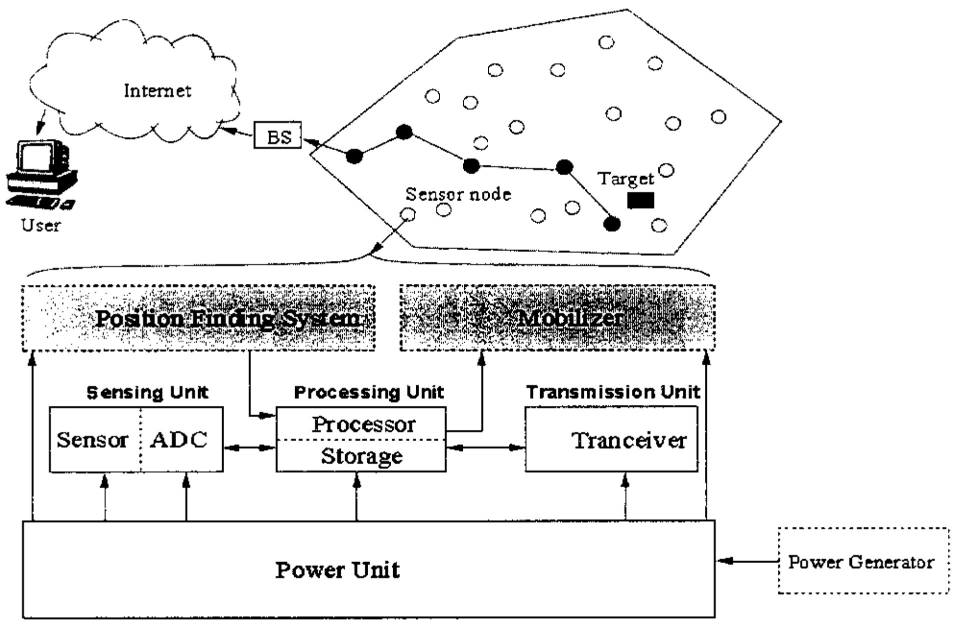


Fig 3.1.1

Components of a sensor node

A sensor node is made up of four major components [5] as shown in the figure 3.1.1 a sensing unit, a processing unit, a transceiver unit and a power unit. They may also have additional application-dependent components such as location finding system, power generator, and mobilizer. Sensing units are usually composed of sensors and analog – to – digital convertors. The processing unit associated with small storage unit manages the procedures that make the sensor collaborate with the other nodes to carry out the assigned sensing tasks. A transceiver unit connects the node to the network. Power unit may be supported by power scavenging units such as solar cells. There are also subunits that are application dependent.

Sensor networks applications

Sensor networks may consist of many different types of sensors such as seismic, low sampling rate magnetic, thermal, visual, infrared, and acoustic and radar, which are able to monitor a wide variety of ambient conditions that include the following:

- Temperature
- Humidity
- Vehicular movement
- Lightning condition
- Pressure
- Soil makeup
- Noise levels
- The presence or absence of certain kinds of objects
- Mechanical stress levels on attached objects
- The current characteristics such as speed, direction, and size of an object.

Sensor nodes can be used for continuous sensing, event detection, event ID, location sensing, and local control of actuators [5]. The concepts of micro-sensing and wireless connection of these nodes promise many new application areas. We categorize the applications into military, environment, health, home and other commercial areas. It is possible to expand this classification with more categories such as space exploration, chemical processing and disaster relief.

Sensor network can use of two types of node [4]

i. Homogeneous - All nodes are of the same type (in terms of battery energy & node hardware)

ii. Heterogeneous – Some nodes are more powerful than others.

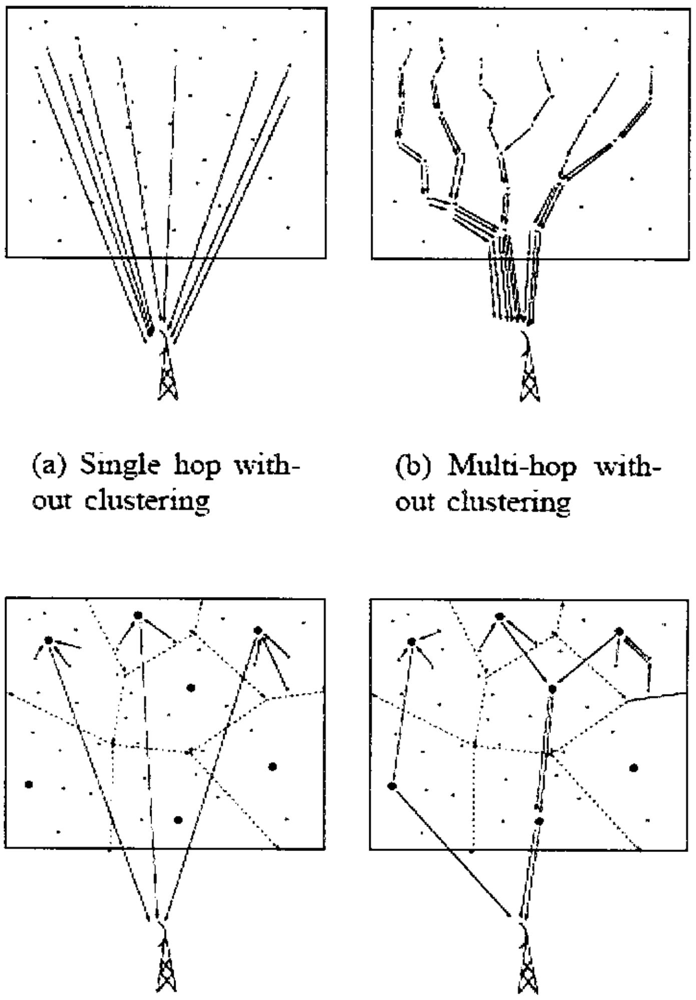
3.2. Clustering in wireless sensor networks:

Network nodes partitioned into several groups. Each group has a cluster head, so that the sensors send raw data to cluster head and the cluster head sends fused data to the processing centre (base station/sink).

Clustering reduces number of nodes contending for channel access, minimizes number of nodes that take part in long distance communication and is used to cover larger area.

Cluster heads work in dual power mode. The head can operate in high power mode for Inter – cluster communication while they use lower power for Intra – cluster communication. Cluster head knows the location of its slave sensors.

In clustered architecture the cluster heads (gateway) form distinct clusters in the system and manage the network in the cluster, perform data fusion to correlate sensor reports and organize sensors by activating a subset relevant to required missions or tasks. The main aim of clustering is to maximize network life time under the constraint of limited power supply.



(a) Single hop without clustering

(b) Multi-hop without clustering

(c) Single hop with clustering

(d) Multi-hop with clustering

Fig 3.2.1

Methods of Clustering

3.2.1 Challenges in Clustering:

- Optimum number of clusters
- Scalability
- Similar size clusters
- Optimum number of overlaps between clusters
- Minimum number of non-clustered nodes
- Minimum communication overhead

3.2.2 Types of Clustering

Following are the different clustering (based on node degree, load & distance etc.) methodologies discussed in the following sections:

i. Static Clustering

ii. Dynamic Clustering

iii. Balanced Clustering

iv. Hierarchical Clustering

3.2.2.1 Static Clustering

Clusters are formed statically at the time of network deployment. The attributes of each cluster (size of cluster, area it covers, members it possesses) are static. The main drawback of this type of clustering is that if a cluster head dies of power depletion, all sensors in the cluster render useless.

3.2.2.2. Dynamic Clustering

In this type of clustering, it is done on demand basis. Sensors do not statically belong to a cluster. They may support different. Clusters can be formed at different times. It is highly adaptive to dynamic environment.

3.2.2.3. Balanced Clustering

Sensor nodes are uniformly distributed around the cluster heads in order to balance the load among heads at the time of clustering, which will improve the life time and minimizes the energy consumption of the system. If the load is not balanced among the different clusters the few heads will be heavily loaded after clustering and will consume their energy very soon. The load handled by cluster head depends on the number of nodes supported by it. A cluster head apart from supporting its members with the radio resources has also to route messages for other nodes belonging to different clusters. Therefore, it is not desirable to have any cluster head overly loaded while some others are lightly loaded. At the same time, it is difficult to maintain a perfectly load balanced system at all times due to frequent detachment and attachment of the nodes from and to the cluster heads.

3.2.2.4. Hierarchical Clustering

More than one level of clustering is used. In this environment, the sensors communicate the gathered data to level-1 cluster heads. The level-1 cluster heads aggregate this data and communicate the aggregated data to level-2 cluster heads and so on [3].

The clustering can also be classified as

i. Physical clustering

ii. Spatial clustering

i. Physical clustering

In this type of clustering, cluster heads are selected periodically according to a hybrid of their residual energy and a secondary parameter, such as node proximity to its neighbors or node degree. Based on this approach we present a protocol, HEED (Hybrid Energy-Efficient Distributed clustering). HEED operates in quasi-stationary networks where nodes are location-unaware and have equal significance

ii. Spatial clustering

Spatial clustering partitions the network into a set of spatial regions with similar observations. Data models are build locally at each node and cluster is formed based on model coefficients (rather than on raw data). These co-efficient are used to capture trends and seasonalties in spatial regions. Scientists can gain insight into the overall distribution patterns and interesting correlations among sensory attributes by grouping such spatial regions which have observed into a single class.

3.3 Heed Protocol: [1]

3.3.1 Clustering Parameters

The overarching goal of our approach is to prolong network lifetime. For this reason, cluster head selection is primarily based on the residual energy of each node. Measuring this residual energy is not necessary, since the energy consumed per bit for sensing, processing, and communication is typically known, and hence residual energy can be estimated. To increase energy efficiency and further prolong network lifetime, we also consider intra-cluster "communication cost" as a secondary clustering parameter. For example, cost can be a function of neighbor proximity or cluster density.

We use the primary clustering parameter to probabilistically select an initial set of cluster heads, and the secondary parameter to "break ties" among them. A tie in this context means that a node falls within the "range" of more than one cluster head. To understand what "range" denotes in this case, observe that a node typically has a number of discrete transmission power levels. Thus, the *cluster range* or *radius* is determined by the transmission power level used for intra-cluster announcements and during clustering. We refer to this level as the *cluster power level*. The cluster power level should be set to one of the lower power levels of a node, to increase spatial reuse, and reserve higher power levels for inter-cluster communication. These higher power levels should cover at least two or more cluster diameters to guarantee that the resulting inter-cluster overlay will be connected. If this condition cannot be satisfied, then our approach for clustering in conjunction with power level selection is inapplicable. The cluster power level dictates the number of clusters in our network. It is non-trivial to determine an optimal cluster power level, because network topology changes due to node failures and energy depletion.

The secondary clustering parameter, intra-cluster communication cost, is a function of (i) cluster properties, such as cluster size, and (ii) whether or not variable power levels are permissible for intra-cluster communication. If the power level used for intra-cluster communication is fixed for all nodes, then the cost can be proportional to (i) node degree, if the requirement is to distribute load among cluster heads, or (ii) node degree, if the requirement is to create dense clusters. This means that a node joins the cluster head with minimum degree to distribute cluster head load (possibly at the expense of increased interference and reduced spatial reuse), or joins the one with maximum degree to create dense clusters. We use the terms *minimum degree cost* and *maximum degree cost* to denote these cost types. Observe that inter-cluster communication is not incorporated in the cost function since local information is insufficient in this case.

3.3.2. Protocol Operation

As discussed in Section 3.1, clustering is triggered every *TCP +TNO* seconds to select new cluster heads. At each node, the clustering process requires a number of iterations, which we refer to as *Niter*. Every step takes time *tc*, which should be long enough to receive messages from any neighbor within the cluster range. We set an initial percentage of cluster heads among all *n* nodes, *Cprob* (say 5%), assuming that an optimal percentage cannot be computed a priori. *Cprob* is only used to limit the initial cluster head announcements, and has no direct impact on the final clusters. Before a node starts executing HEED, it sets its probability of becoming a cluster head, *CHprob*, as follows:

$$CHprob = Cprob \times Eresidual / Emax$$

where *Eresidual* is the estimated current residual energy in the node, and *Emax* is a reference maximum energy (corresponding to a fully charged battery), which is typically identical for all nodes. The *Chprob* value of a node, however, is not allowed to fall below a certain threshold *pmin* , that is selected to be inversely proportional to *Emax*. This restriction is essential for terminating the algorithm in *Niter* = *O*(1) iterations, as we will show later. Observe that our clustering approach is capable of handling heterogeneous node batteries. In this case, every node will have its own *Emax* value. During any iteration *i*, *Niter*, every "uncovered" node elects to become a cluster head with probability *CHprob*. After step *i*, the set of tentative cluster heads, *SCH*, is set to *{*cluster heads after step *i* − 1 *[* new heads selected in step *i}*. A node *vi* selects its cluster head (*my cluster head*) to be the node with the lowest cost in *SCH* (*SCH* may include *vi* itself if it is selected as a tentative cluster head). Every node then doubles its *CHprob* and goes to the next step. Note that if different power levels can be used for intra-cluster communication, then line 1 in phase I must be modified as follows: Discover neighbors within every power level *Pwri* _ *Pwrc*, where *Pwrc* is the cluster power level. In this case only, we assume that if cluster head *u* can reach a node *v* with power level *l*, then *v* can reach *u* with level *l* as well. Neighbor discovery is not necessary every time clustering is triggered. This is because in a stationary network, where nodes do not die unexpectedly, the neighbor set of every node does not change very frequently. In addition, HEED distribution of energy consumption extends the lifetime of all the nodes in the network, which adds to the stability of the neighbor set. Nodes also automatically update their neighbor sets in multi-hop networks by periodically sending and receiving heartbeat messages.

Note also that if a node elects to become a cluster head, it sends an announcement message *cluster head msg(Node ID, selection status, cost)*, where the selection status is set to *tentative CH*, if its *CHprob* is less than 1, or *final CH*, if its *CHprob* has reached 1. A node considers itself "covered" if it has heard from either a *tentative CH* or a *final CH*. If a node completes HEED execution without selecting a cluster head that is *final CH*, it considers itself uncovered, and announces itself to be a cluster head with state *final CH*. A *tentative CH* node can become a regular node at a later iteration if it finds a lower cost cluster head. Note that a node can elect to become a cluster head at consecutive clustering intervals if it has high residual energy and low cost.

3.3.3 Pseudo Code

```
Initialize
 Snbr  ← {v: v lies within my cluster range}
 Compute and broadcast cost to E Snbr
 CHprob ← max(Cprob × Eresidual/Emax , pmin)
 is final CH ← FALSE
Repeat
      If ((SCH ← {v: v is a cluster head})!= NULL)
       my cluster head ← least cost(SCH)
          If (my cluster head = NodeID)
            If (CHprob = 1)
              Cluster head msg(NodeID, final CH,cost)
              is final CH ← TRUE
           Else
             Clusterhead msg(NodeID, tentative CH,cost)
        ElseIf (CHprob = 1)
           Cluster head msg(NodeID,final CH,cost)
           is final CH ← TRUE
       ElseIf Random(0,1) <= CHprob
         Cluster head msg(NodeID,tentative CH,cost)
      CHprevious ← CHprob
      CHprob ← min(CHprob × 2, 1)
Until CHprevious = 1
Finalize
 If (is final CH = FALSE)
    If ((SCH ← {v: v is a final cluster head}) !=NULL)
            my cluster head ← least cost(SCH)
            join cluster(cluster head ID, NodeID)
    Else Cluster head msg(NodeID, final CH, cost)
 Else Cluster head msg(NodeID, final CH, cost)
```

3.3.4 Results

The following table gives the details of the number of rounds a cluster can go through before the first node expires and until 50% of the nodes expires. The results are tabulated for different topologies involving different number of nodes.

Table 3.3.4.1: Lifetime of HEED protocol

No of Nodes	No of rounds before the Ist node expires	No of rounds until 50% of nodes expires
50	216	1111
60	246	1320
70	232	1331
80	224	1415
90	220	1393

As seen from the table as the number of nodes increases the lifetime of the network also increases. However the number of rounds before the first node expires varies irrespective of the number of nodes in the topology. This is because the nodes are randomly deployed and no specific positions of the nodes are assumed. The results of the table are depicted in a graphical format below,

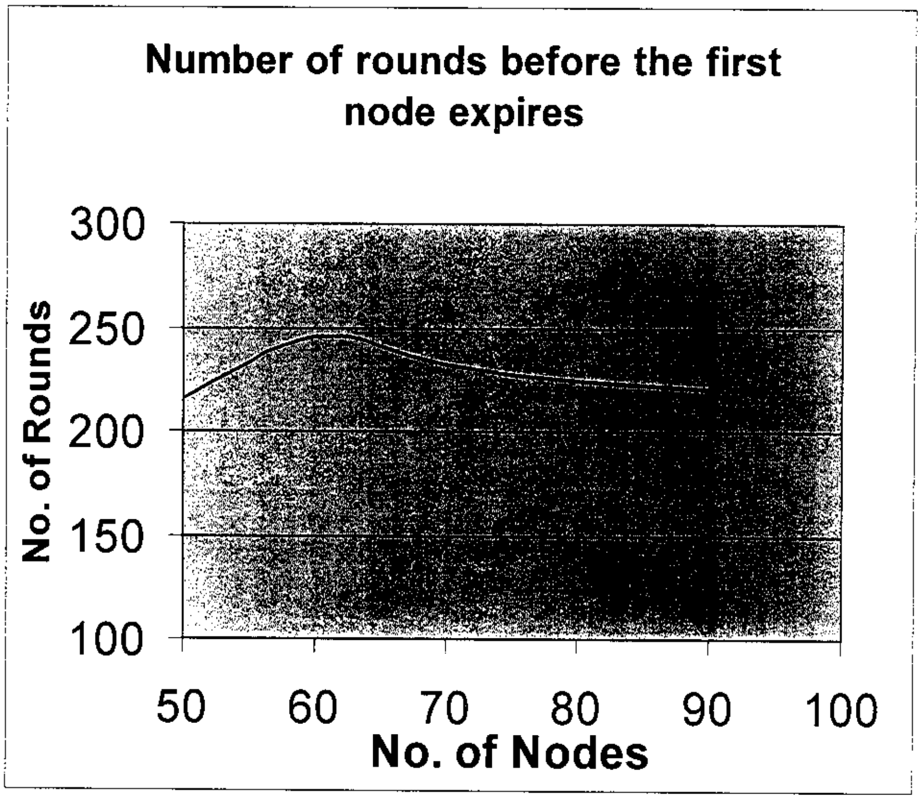


Fig 3.3.4.1

Node Lifetime

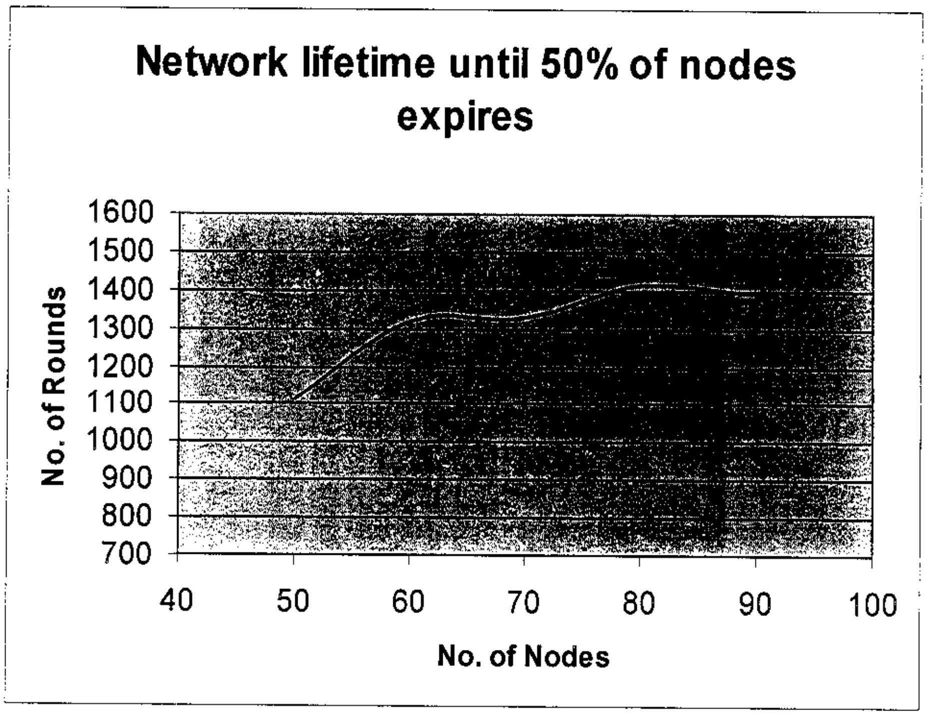


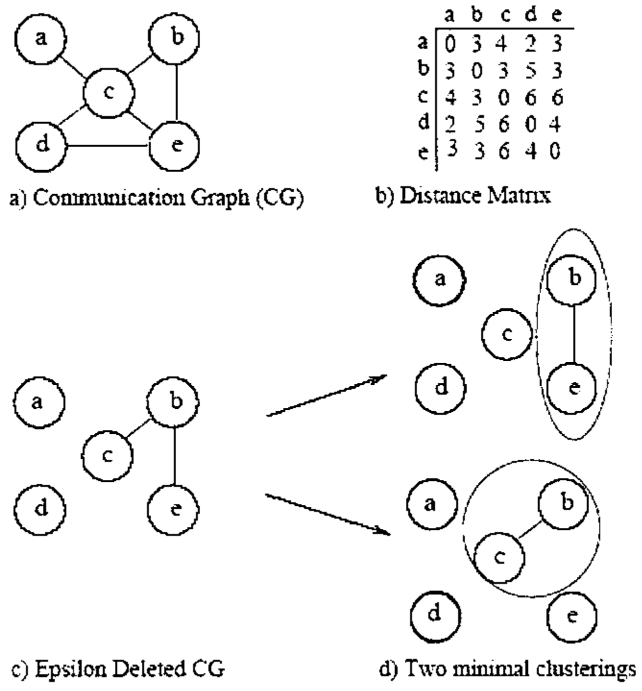
Fig 3.3.4.2

Network Lifetime

3.4. Spatial Clustering:

3.4.1. Introduction [2]

A good spatial clustering algorithm should group nodes in the sensor network based on data semantics. In order to cluster based on data patterns rather than on raw data, we regress data to build data models. The model coefficients are used as the *features* from each node. A sensor node i processes local data to extract feature Fi. We capture the (dis) similarity between two features Fi and Fj by distance d(Fi ; Fj). For example, consider the communication graph CG of a sensor network S as shown in Figure 3.4.1a). Figure 3.4.1b) shows an example of a distance matrix which measures the feature similarity of nodes in S. Clustering algorithms should identify the boundaries of spatial clusters. Two nodes that are communication neighbors should not cluster using a communication edge if the dissimilarity between their data features exceeds a threshold. We denote this threshold by *E*. Removing all such edges from CG results in an *Epsilon Deleted CG*. For $E = 3$, the Epsilon Deleted Communication Graph of the example is shown in Figure 3.4.1c). We see that edge eac is deleted from CG since d(Fa; Fc) = 4 > 3. For the same reason, we see that edges ecd, ece and ede are deleted. Neighbors in this graph are referred to as *E*- satisfying neighbors. For example node b is an *E*-satisfying neighbor of nodes c and e. The *E* parameter captures *local* dissimilarities. We use the parameter to capture *global* dissimilarity. Two nodes should not be grouped in the same cluster if the distance between their features exceeds ∂. If $\partial = 5$, then node c and node e cannot belong to the same cluster since d(Fc; Fe) = 6 > 5.



	a	b	c	d	e
a	0	3	4	2	3
b	3	0	3	5	3
c	4	3	0	6	6
d	2	5	6	0	4
e	3	3	6	4	0

Fig 3.4.1

Epsilon Deleted Graph

Given a set of sensors S, their communication graph CG, their distance matrix D, and two real numbers E and ∂, a set of sensors C is called a E -∂ Cluster with respect to (E;∂) if the following two conditions hold.

1. The communication sub graph induced by C on the Epsilon deleted CG is connected.

2. For every pair of nodes i and j belonging to C, d(Fi ; Fj) <=∂. We refer to this as the ∂ condition or ∂-compactness.

Clustering is defined as the partition of the communication graph CG into a set of *disjoint* E -∂ clusters.

For $E = 3$ and $\partial = 5$, the two possible minimal clustering for the *Epsilon-deleted Graph* shown in Figure 3c) are shown in Figure 3d). One corresponds to the set of clusters ({a}, {c}, {d}, {b, e}) whereas the other corresponds to ({a}, {d}, {e}, {b, c}).

3.4.2 Feature modeling and distance

In order to find underlying spatial patterns in a sensor network, raw data from all the sensors can be transmitted to the centralized base station, and the base station can mine the data. But this would incur a huge communication overhead. An interesting alternative is to build a *local* data model at each node that captures the underlying data structure and transmit only the model coefficients. Clustering based on model coefficients rather than on raw data has two very important advantages

a) It discovers spatial regions with similar trends and periodicities in data and

b) It decreases the communication overhead by an order of magnitude.

The correlation analysis tool measures the relationship between two data sets that are scaled to be independent of the unit of measurement. The population correlation calculation returns the covariance of two data sets divided by the product of their standard deviations based on the following formulas.

$$\rho_{x,y} = \mathrm{cov}(x,y)/\sigma_x * \sigma_y \longrightarrow 1$$

$$\sigma^2_x = 1/n\ \Sigma(x_i - \mu_x)^2 \longrightarrow 2$$

$$\sigma^2_y = 1/n\ \Sigma(y_i - \mu_y)^2 \longrightarrow 3$$

The correlation analysis tool is used to determine whether two ranges of data move together — that is, whether large values of one set are associated with large values of the other (positive correlation), whether small values of one set are associated with large values of the other (negative correlation), or whether values in both sets are unrelated (correlation near zero).

3.4.3 TAO Project

TAO (Tropical Atmosphere Ocean) Project is used to collect real time data from moored ocean buoys for improved deduction, understanding and prediction of El Nino and La Nina [6].

El Niño can mean severe droughts and deadly forest fires. Ecuadorians, Peruvians, or Californians, on the other hand, associate it with lashing rainstorms that can trigger devastating floods and mudslides. Severe El Niño events have resulted in a few thousand deaths worldwide, left thousands of people homeless, and caused billions of dollars in damage. Yet residents on the northeastern seaboard of the United States can credit El Niño with milder-than-normal winters (and lower heating bills) and relatively benign hurricane seasons.

Originally, the name El Niño (Spanish for "the Christ child") [8] was coined in the late 1800s by fishermen along the coast of Peru to refer to a seasonal invasion of warm southward ocean current that displaced the north-flowing cold current in which they normally fished; typically this would happen around Christmas. Today, the term no longer refers to the local seasonal current shift but to part of a phenomenon known as El Niño-Southern Oscillation (ENSO), a continual but irregular cycle of shifts in ocean and atmospheric conditions that affect the globe. El Niño has come to refer to the more pronounced weather effects associated with anomalously warm sea surface temperatures interacting with the air above it in the eastern and central Pacific Ocean. Its counterpart--effects associated with colder-than-usual sea surface temperatures in the region--was labeled "La Niña" (or "little girl") as recently as 1985.

The shift from El Niño conditions to La Niña and back again takes about four years. Understanding this irregular oscillation and its consequences for global climate has become possible only in recent decades as scientists began to unravel the intricate relationship between ocean and atmosphere. Although meteorologists have long been forecasting daily weather based on atmospheric measurements taken around the world, they had relatively little information about conditions in many parts of the world's oceans until the advent of arrays of fixed unmanned mid ocean buoys in the Pacific Ocean and orbiting satellites.

But technological advances were not the only key. As the following article recounts, atmospheric and oceanographic researchers, after years of independent inquiry into the basic workings of air and sea, at last joined forces. An elegant synthesis of these two fields of research now enables climatologists and oceanographers to construct theoretical models to simulate and predict the broad climate changes associated with ENSO. For example, scientists can now warn vulnerable populations of an impending El Niño event several months in advance, providing precious time in which to take steps to mitigate its worst effects. Invaluable as this prediction of El Niño is, it is just the first step toward the much longer-term goal of providing the climatic counterpart to the daily weather prediction that we have come to take for granted.

The data is gathered from the TAO project official web site [7]

Table 3.4.3.1: TAO Data

Hour/Sensor	1	2	3	4	5
0	23.047	24.049	22.977	23.783	29.291
1	23.094	24.002	23.001	23.664	29.429
2	23.026	24.021	23.014	23.64	29.511
3	22.998	24.009	23.033	23.621	29.53
4	22.965	23.962	23.054	23.648	29.443
5	22.95	23.901	23.086	23.624	29.383
6	22.914	23.885	23.162	23.608	29.323
7	22.947	23.958	23.189	23.569	29.247
8	22.99	24.006	23.166	23.597	29.263
9	22.941	23.969	23.204	23.59	29.187
10	23.005	23.952	23.199	23.597	29.129
11	23.006	23.978	23.187	23.579	29.121
12	23.048	23.933	23.191	23.584	29.143
13	23.05	23.989	23.175	23.582	29.15
14	23.056	23.989	23.17	23.59	29.106
15	23.048	23.992	23.178	23.595	29.084
16	23.065	23.995	23.202	23.645	29.067
17	23.088	24.02	23.228	23.677	29.084
18	23.156	24.071	23.27	23.698	29.072
19	23.195	24.074	23.432	23.806	29.066
20	23.313	24.17	23.494	23.948	29.064
21	23.42	24.313	23.55	24.026	29.076
22	23.41	24.383	23.606	24.062	29.102
23	23.448	24.472	23.631	24.105	29.127

Hour/Sensor	6	7	8	9	10	11
0	26.612	24.677	26.628	26.315	29.138	30.271
1	26.573	24.676	26.650	26.272	29.141	30.627
2	26.565	24.610	26.624	26.223	29.130	30.712
3	26.523	24.587	26.606	26.219	29.141	30.698
4	26.526	24.583	26.569	26.197	29.141	30.571
5	26.479	24.571	26.572	26.182	29.123	30.337
6	26.477	24.548	26.542	26.179	29.109	30.242
7	26.457	24.555	26.536	26.177	29.099	30.168
8	26.433	24.541	26.536	26.168	29.107	30.085
9	26.438	24.530	26.542	26.149	29.089	30.047
10	26.426	24.517	26.552	26.172	29.087	30.045
11	26.424	24.533	26.544	26.182	29.087	29.958
12	26.403	24.527	26.545	26.200	29.089	29.908
13	26.389	24.527	26.546	26.197	29.081	29.890
14	26.388	24.534	26.566	26.190	29.081	29.862
15	26.377	24.539	26.545	26.174	29.074	29.821
16	26.372	24.575	26.566	26.167	29.076	29.783
17	26.365	24.590	26.573	26.163	29.091	29.781
18	26.378	24.690	26.596	26.179	29.108	29.812
19	26.396	24.776	26.635	26.196	29.118	29.831
20	26.413	24.844	26.677	26.231	29.148	29.850
21	26.427	24.907	26.729	26.252	29.201	30.017
22	26.441	24.966	26.761	26.275	29.273	30.054
23	26.459	24.983	26.827	26.297	29.340	30.232

Table 3.4.3.2: Sensor Locations

1	Location: 0N 125W
2	Location: 0N 140W
3	Location: 2N 125W
4	Location: 2S 125W
5	Location: 2S 180W
6	Location: 5N 140W
7	Location: 5S 125W
8	Location: 5S 140W
9	Location: 8S 125W
10	Location: 8S 155W
11	Location: 8S 170W

3.4.4 Spanning tree clustering

The spanning tree based algorithm consists of two phases. In the first phase, the algorithm picks up an arbitrary node as a root and builds a spanning tree (BFS, DFS, etc) on the *Epsilon Deleted Communication Graph* of the network. In the second phase *∂-sub tree clustering*, we partition the tree into sub trees which satisfy the ∂-condition. The second phase of the algorithm is executed at each node is as follows. Every node initializes its *height* to 0. Beginning with the leaves, each leaf sends its *height* (0) and its feature Fi to its parent p. Each parent node maintains its own *height*, its local feature Fp, and the identifier of the child with the maximal height in the variable *highest child*. When it receives a new height h from one of its children, it updates h to h + d(Fi; Fp) and then checks if h + height exceeds ∂. If it does, then node p instructs the child whose height is the largest (*highest child*) to detach. Otherwise, both the *height* and *highest child* are updated.

After receiving the heights of all its children, p sends its own height and feature Fp to its parents. Every detached *highest child* with its sub tree forms a new cluster with the *highest child* as the root. The pseudo code for this algorithm is given below. The experimental results for the spanning tree algorithm are averaged over different root positions. Consider the example shown in Figure 3.4.4.1 for $\partial = 4$. We use the *Epsilon-Deleted Graph* shown in Figure 3.4.4.1a) as a running example for the clustering algorithms. The label on each edge eij shown in this figure is the feature distance d(Fi ; Fj) between neighbors i and j.

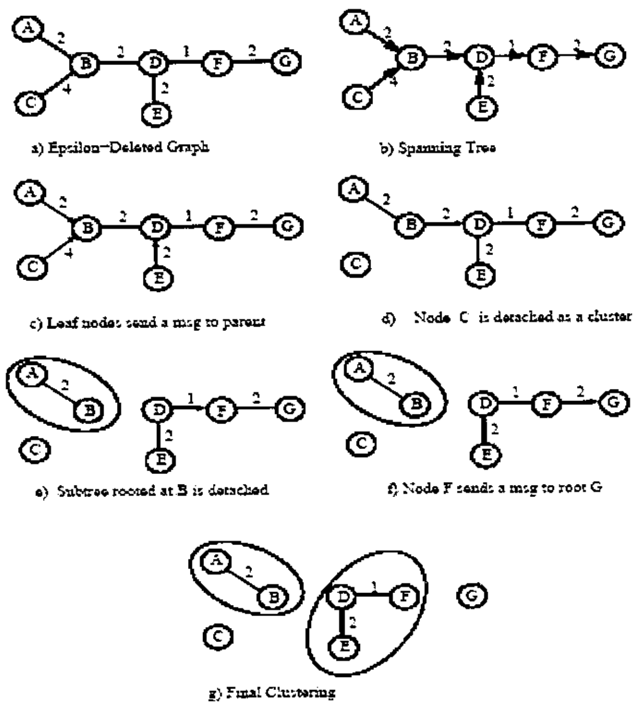

a) Epsilon-Deleted Graph
b) Spanning Tree
c) Leaf nodes send a msg to parent
d) Node C is detached as a cluster
e) Subtree rooted at B is detached
f) Node F sends a msg to root G
g) Final Clustering

Fig 3.4.4.1

Spanning Tree Clustering

Now, we show the execution trace of the algorithm on this example. Assume that the spanning tree returned in the first phase of the algorithm is as shown in Figure 3.4.4.1b). The second phase *∂-subtree-clustering* starts in Figure 3.4.4.1c). Every node i's height variable is initialized to 0. Leaf nodes A, C and E propagate their heights' (0) to their parents B and D as shown in c). Assume node B receives node A's message earlier than node C. Node B updatesA's heightA to 0 + 2 = 2 and sets its highest child to node A and updates height B to 2. Now, node C sends its heightC to B. Node B updates C's height to 0 + 4 = 4. Since 4 + 2 > 4 = ∂), it detaches the child with the maximum height which is node C. Node C forms a singleton cluster as shown in Figure 3.4.4.1d). Since nodeB has received the messages from both its children, it propagates its heightB = 2 to its parent node D. This process is carried on as shown finally resulting in the clustering as shown in Figure 3.4.4.1g).

P- 1636

3.4.5 Pseudo Code

```
∂-subtree clustering at node p (0 <= p <= N)
receive(height : h;Feature : Fi)i
h = h + D(Fp; Fi) // Fp is the local feature
if((h + height) > ∂))
    // must detach some child
    if(h > height)
        Detach subtree rooted at i as a cluster.
    else
        Detach subtree rooted at highest child as a cluster.
        height = h
        highest child = i
else if(h > height)
    // update i as the highest child.
    height = h
    highest child = i
send(height : h;Feature : Fp)i
    Precondition: p has received messages from all children.
                  Immediately satisfied at a leaf.
```

3.4.6 Results

The tables below give the results of the spanning tree algorithm with varying δ and ε values.

Table 3.4.6.1: *E*- δ Analysis

$E(\delta=1.2)$	No of clusters	$\delta(E=0.9)$	No of clusters
1	5	3	3
0.9	6	2	4
0.82	7	1.7	5
0.8	8	1	6
0.7	9	0.9	7

As seen from the table as *E* value decreases the number of clusters increase. This is evident as the threshold of the local dis-similarities (*E*) is reduced the number of clusters are bound to increase. But in the case of δ, which is the global dis-similarity parameter, as the value increases the number of clusters formed will decrease. The results of the tabulation are depicted in a graph below,

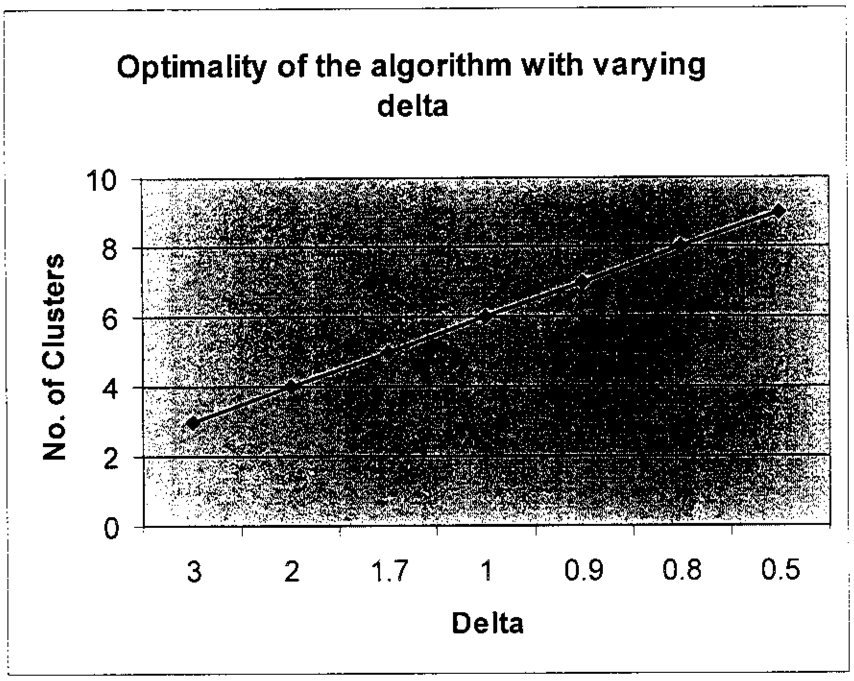


Fig 3.4.6.1

Delta Analysis

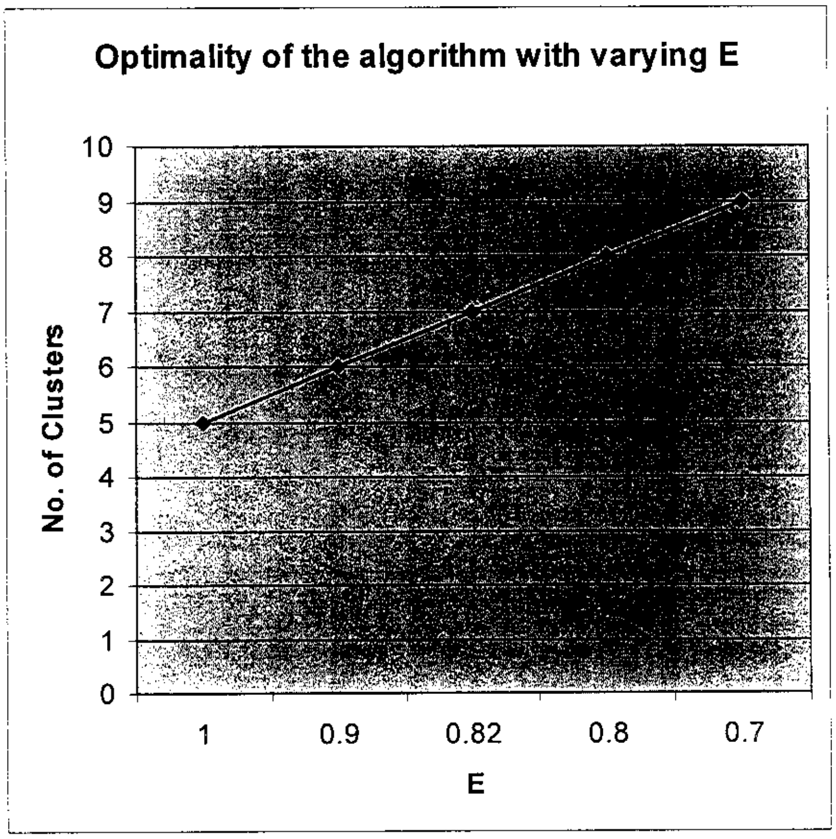
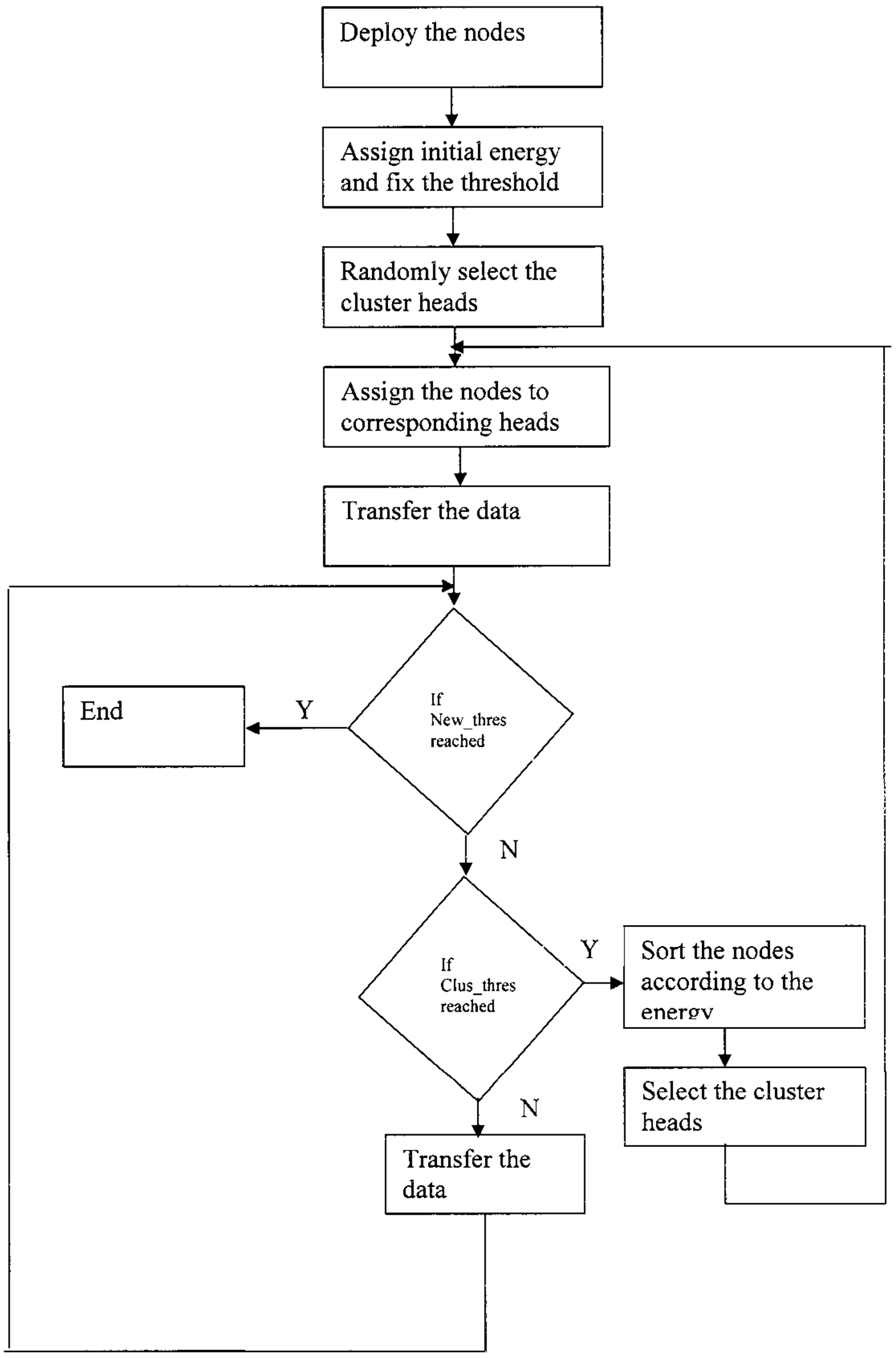


Fig 3.4.6.2

Epsilon Analysis

4. DESIGN DOCUMENTS:

4.1 Flow Diagram


Deploy the nodes
Assign initial energy and fix the threshold
Randomly select the cluster heads
Assign the nodes to corresponding heads
Transfer the data
If New_thres reached
Y
End
N
If Clus_thres reached
Y
Sort the nodes according to the energy
Select the cluster heads
N
Transfer the data

5. TESTING

Testing is done to detect the errors in the software. This implies not only to coding phase but to uncover errors introduced in all the previous phases.

The following were the types of tests that were performed

Unit Testing

Each and every module is tested separately to check if its intended functionality is met. Some of the unit tests that were performed were,

- To check if the nodes were being deployed properly.
- Check if the sensors were able to transmit the data to other nodes.
- To check if the energy levels of the sensors were proper
- The clustering is initiated at appropriate threshold levels

Integrated Testing

It is the testing performed to detect errors on interconnection between modules. Here, the process of sub-dividing the task and then sending it to other module and check if the required task by the module is performed. All these events when done together must satisfy the purpose

System Testing

The system is tested against the system requirements to see if all the requirements are met and if the system performs as per the specified requirements. The system is tested as a whole to check for its functionality.

6. FUTURE ENHANCEMENTS:

In the present work the spatial clustering protocol uses a single parameter to cluster the sensors. We intend to use multiple parameters to perform the same clustering. We also intend to work on trend analysis. Trend analysis is the process of predicting the future values from the sensors with the help of the model co-efficients. The simulated results obtained can be used in a number of real time applications like seismic monitoring, oceanography, environmental monitoring etc.,

7. CONCLUSION

We have presented an energy-efficient distributed clustering approach for ad-hoc sensor networks. Our approach is hybrid: cluster heads are randomly selected based on their residual energy, and nodes join clusters such that communication cost is minimized. Based on this approach, we have implemented the HEED protocol, which terminates in a constant number of iterations, independent of the network diameter. HEED operates in quasi-stationary networks where nodes are location-unaware and have equal significance. Simulation results show that HEED prolongs network lifetime, and the clusters it produces exhibit several appealing characteristics. HEED parameters, such as the minimum selection probability and network operation interval, can be easily tuned to optimize resource usage according to the network density and application requirements. HEED can also be useful in multi-hop networks if the necessary conditions for connectivity (the relation between cluster range and transmission range under a specified density model) hold. Our approach can be applied to the design of several types of sensor network protocols that require energy efficiency, scalability, prolonged network lifetime, and load balancing.

We have considered the important problem of spatial clustering in sensor networks. We built data models at each node to capture trends in data and reduce communication costs. We presented distributed algorithms for *in-network* clustering to generate clusters with a low communication overhead. We answer range queries efficiently based on the δ-compact clusters. The *in-network* clustering and modeling reduced the communication cost by two orders of magnitude. Clustering based on spatial models help scientists and engineers gain valuable insight into the trends and seasonalities of the underlying data. The query cost involved in spatial clustering is less as the model co-efficient answers most of the queries itself. This kind of spatial clustering proves suitable for many sensor network applications such as seismic monitoring, oceanography, environmental monitoring etc., The approach of clustering sensor nodes based on these models is promising

8. APPENDIX

8.1 Sample Code

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<complex.h>
#include<math.h>
#include<stdlib.h>    // randomize
#include<graphics.h>

#define No 60
#define Co 6
struct node
{
long double d[10],l[10];

int x1,y1;
int node_color;
int clusprob;
double life_thres;
int cluster_head;   //0 ordinary node
int node_id;
int is_head;
int is_life;
long double residual;
long double ener_drain;
}nodes[No];

struct cluster
{
int x2,y2;
int cluster_id;
double clus_thres;
int color;
long double dis_base;
int node_count;
long double temp112;
long double residual;

}clust[Co];

int count_round_head=0;
int no_of_rounds=0;
int a=0,cit=0;
```

```
long double er=0;
long double eft=0,et=0,ech=0;
long double eamp=5.46*pow(10,-12);
long double elec=50*pow(10,-9);
long double ef=.021*pow(10,-9);
long double etx=0,secterm=0;
int N=0;
int k1=2;
int counter=0;
int counter11=1,counter12=1;
long double resi_temp[No];
long double tem=0;
long double temp_ech[Co],temp_drain[No];
int nn=0;
void clust_to_white();
int c1=275;
int c2=225;

void main()
{
int gd=DETECT,gm=0;//graphics mode changed from text mode
initgraph(&gd,&gm,"c:\\tc");
randomize();
int x=0,y=0;
int temp;

for(int i=0;i<Co;i++)
{
temp_ech[i]=1;
}

for(i=0;i<No;i++)
{
temp_drain[i]=1;
}

for(i=0;i<No;i++)
{
nodes[i].is_life=1;
}

for(i=0;i<No;i++)
{
x=random(550);
y=random(450);
nodes[i].x1=x;
nodes[i].y1=y;
circle(x,y,2);
}
getch();
setcolor(0);
circle(nodes[0].x1,nodes[0].y1,2);

//base station
setcolor(7);
setfillstyle(1,7);
fillellipse(c1,c2,7,7);

//formation of cluster head  for the first round
for(int k=0;k<Co;k++)
{
clust[k].cluster_id=k;
clust[k].x2=nodes[k].x1;
clust[k].y2=nodes[k].y1;
setcolor(25+k);
clust[k].color=25+k;
a=25+k;
setfillstyle(1,a);
fillellipse(clust[k].x2,clust[k].y2,5,5);
}

//DISTANCE CALCULATION
//separation of cluster heads

for(k=0;k<No;k++)
{
nodes[k].is_head=0;
}
for(k=0;k<No;k++)
{
        for(int l=0;l<Co;l++)
        {
         if((clust[l].x2==nodes[k].x1)&&(clust[l].y2==nodes[k].y1))
         {
          nodes[k].is_head=1;
         }
        }
}

//distance calculated for reamining nodes

long double temp1,temp2,temp3;
```

```
for(k=0;k<No;k++)
{
        if(nodes[k].is_head!=1)
        {
            temp1=pow((nodes[k].x1-clust[0].x2),2);
            temp2=pow((nodes[k].y1-clust[0].y2),2);
            temp3=temp1+temp2;
            nodes[k].d[0]=sqrt(temp3);
            nodes[k].l[0]=nodes[k].d[0];

            temp1=pow((nodes[k].x1-clust[1].x2),2);
            temp2=pow((nodes[k].y1-clust[1].y2),2);
            temp3=temp1+temp2;
            nodes[k].d[1]=sqrt(temp3);
            nodes[k].l[1]=nodes[k].d[1];

            temp1=pow((nodes[k].x1-clust[2].x2),2);
            temp2=pow((nodes[k].y1-clust[2].y2),2);
            temp3=temp1+temp2;
            nodes[k].d[2]=sqrt(temp3);
            nodes[k].l[2]=nodes[k].d[2];

            temp1=pow((nodes[k].x1-clust[3].x2),2);
            temp2=pow((nodes[k].y1-clust[3].y2),2);
            temp3=temp1+temp2;
            nodes[k].d[3]=sqrt(temp3);
            nodes[k].l[3]=nodes[k].d[3];

            temp1=pow((nodes[k].x1-clust[4].x2),2);
            temp2=pow((nodes[k].y1-clust[4].y2),2);
            temp3=temp1+temp2;
            nodes[k].d[4]=sqrt(temp3);
            nodes[k].l[4]=nodes[k].d[4];

            temp1=pow((nodes[k].x1-clust[5].x2),2);
            temp2=pow((nodes[k].y1-clust[5].y2),2);
            temp3=temp1+temp2;
            nodes[k].d[5]=sqrt(temp3);
            nodes[k].l[5]=nodes[k].d[5];
        }
}

//color assigining to nodes based on minimum distance

long double temp4;
for(int v=0;v<No;v++)
```

```
   {
    for(int b=0;b<Co;b++)
    {
     for(int m=b+1;m<Co;m++)
      {
          if(nodes[v].is_head==0)
           {
            if((nodes[v].d[b])>(nodes[v].d[m]))
             {
                  temp4=nodes[v].d[b];
                  nodes[v].d[b]=nodes[v].d[m];
                  nodes[v].d[m]=temp4;
             }
           }
      }
    }
   }

 //sorted dist for a particular node
 int m;
 for(k=0;k<No;k++)
  {
   for(m=0;m<Co;m++)
   {
    if(nodes[k].is_head==0)
    {
     if((nodes[k].l[m])==(nodes[k].d[0]))
      {
          setcolor(25+m);
          circle(nodes[k].x1,nodes[k].y1,2);
          nodes[k].node_color=25+m;
      }
    }
   }
  }
 getch();

 //reelection
 int flag_node_die=0;
 int count_tot_life=0;
 while(flag_node_die==0)
 {
 count_tot_life++;

 //counting the number of nodes
```

```
int
count1=0,count2=0,count3=0,count4=0,count5=0,count6=0,count7=0,count8=
0,count9=0;
for(int k=0;k<Co;k++)
{
clust[k].node_count=0;
}


for(k=0;k<No;k++)
 {
   switch(nodes[k].node_color)
   {
    case 25:
        count1++;
        break;
    case 26:
        count2++;
        break;
    case 27:
        count3++;
        break;
    case 28:
        count4++;
        break;
    case 29:
        count5++;
        break;
    case 30:
        count6++;
        break;
   }
 }
clust[0].node_count=count1;
clust[1].node_count=count2;
clust[2].node_count=count3;
clust[3].node_count=count4;
clust[4].node_count=count5;
clust[5].node_count=count6;

//distance calculation for the head frm base

long double temp1=0,temp2=0,temp3=0;
 for(k=0;k<Co;k++)
 {
   clust[k].dis_base=0;
 }
 for(k=0;k<Co;k++)
 {
 temp1=pow((clust[k].x2-c1),2);
 temp2=pow((clust[k].y2-c2),2);
 temp3=temp1+temp2;
 clust[k].dis_base=sqrt(temp3);
}

//energy initialization for nodes...

//cluster head calculation
long double d1=0;
if(counter11==0)
{
for(k=0;k<Co;k++)
{
temp_ech[k]=clust[k].residual;
}
}
counter11=0;
for(k=0;k<Co;k++)
{
N=clust[k].node_count;
d1=clust[k].dis_base;
er=elec*4000*N;
eft=ef*4000*N;
et=(elec + (eamp*pow(d1,k1)))*4000;
ech=er+eft+et;
clust[k].temp112=ech;
temp_ech[k]=temp_ech[k]-clust[k].temp112;
ech=0;
}
long double d2=0;
if(counter12==0)
{
  for(k=0;k<No;k++)
  {
   if(nodes[k].is_head==0)
   {
    temp_drain[k]=nodes[k].residual;
   }
  }
}
counter12=0;
for(k=0;k<No;k++)
```

```
{
  if(nodes[k].is_head==0)
  {
   d2=nodes[k].d[0];
   secterm=eamp*(pow(d2,k1));
   etx=(elec+secterm)*4000;
   nodes[k].ener_drain=etx;
   temp_drain[k]=temp_drain[k]-nodes[k].ener_drain;
   etx=0;
  }
}
long double ss=0;
long double ss1=0;
int flag=0;
counter=0;
while((temp_ech[0]>(50*pow(10,-12)))&&(temp_ech[1]>(50*pow(10,-
12)))&&(temp_ech[2]>(50*pow(10,-12)))&&(temp_ech[3]>(50*pow(10,-
12)))&&(temp_ech[4]>(50*pow(10,-12)))&&(temp_ech[5]>(50*pow(10,-
12)))&&flag==0)
{
  for(k=0;k<Co;k++)
  {
   ss= temp_ech[k]-clust[k].temp112;
   if(ss<0)
   {
     flag=1;
   }
   else
   {
     temp_ech[k]=ss;
   }
  }
  count_round_head++;
  for(k=0;k<No;k++)
  {
   if(nodes[k].is_head==0)
   {
    ss1=temp_drain[k]-nodes[k].ener_drain;
    if((ss1<0)&&nodes[k].is_life==1)
     {
        nodes[k].is_life=0;
        cit++;
        if(cit==1)
         {
          printf(" nn=%d ",nn);
         }
```

```
          if(cit>(No/2))
          {
           flag_node_die=1;
          }
     }
    else
    {
       temp_drain[k]=ss1;
    }
   }
  }
  nn++;
 }
 for(k=0;k<Co;k++)
 {
  clust[k].residual=temp_ech[k];
 }
 for(k=0;k<No;k++)
 {
  if(nodes[k].is_head==0)
  {
   nodes[k].residual=temp_drain[k];
  }
 }

 clust_to_white();
 //reelection
 //conversion of heads to ordinary nodes
 int z=0;
 for(k=0;k<No;k++)
 {
   if(nodes[k].is_head==1)
   {
    nodes[k].x1=clust[z].x2;
    nodes[k].y1=clust[z].y2;
    nodes[k].residual=clust[z].residual;
    z++;
   }
 }
 //sorting

 for(k=0;k<No;k++)
 {
  resi_temp[k]=0;
 }
 for(k=0;k<No;k++)
```

```
{
resi_temp[k]=nodes[k].residual;
}

for(k=0;k<No;k++)
 {
  for(int p=k+1;p<No;p++)
  {
   if((resi_temp[k])<(resi_temp[p]))
   {
     tem=resi_temp[k];
     resi_temp[k]=resi_temp[p];
     resi_temp[p]=tem;
   }
  }
 }
int posx[Co];
int qq=0;
for(k=0;k<Co;k++)
{
posx[k]=0;
}
 for(k=0;k<No;k++)
 {
   for(int l=0;l<Co;l++)
   {
    if(nodes[k].residual== resi_temp[l])
    {
      posx[qq]=k;
      qq++;
    }
   }
 }
 for(k=0;k<Co;k++)
 {
  clust[k].x2=0;
  clust[k].y2=0;
  clust[k].residual=0;
 }
 int pp=0,kk=0;
 for(qq=0;qq<Co;qq++)
 {
  kk=posx[qq];
  clust[pp].residual=nodes[kk].residual;
  clust[pp].x2=nodes[kk].x1;
  clust[pp].y2=nodes[kk].y1;
 pp++;
}

//change all nodes to white
setcolor(7);
for(k=0;k<No;k++)
{
 circle(nodes[k].x1,nodes[k].y1,2);
}
getch();
for(k=0;k<Co;k++)
{
 setcolor(25+k);
 clust[k].color=25+k;
 a=25+k;
 setfillstyle(1,a);
 fillellipse(clust[k].x2,clust[k].y2,5,5);
}
getch();

//distance calculation
for(k=0;k<No;k++)
{
 nodes[k].is_head=0;
}
for(k=0;k<No;k++)
{
        for(int l=0;l<Co;l++)
        {
        if((clust[l].x2==nodes[k].x1)&&(clust[l].y2==nodes[k].y1))
        {
         nodes[k].is_head=1;
        }
        }
}
//distance calculated for reamining nodes
temp1=0;
temp2=0;
temp3=0;
for(k=0;k<No;k++)
{
        if(nodes[k].is_head!=1)
        {
            temp1=pow((nodes[k].x1-clust[0].x2),2);
            temp2=pow((nodes[k].y1-clust[0].y2),2);
            temp3=temp1+temp2;
```

```
                nodes[k].d[0]=sqrt(temp3);
                nodes[k].l[0]=nodes[k].d[0];
                temp1=pow((nodes[k].x1-clust[1].x2),2);
                temp2=pow((nodes[k].y1-clust[1].y2),2);
                temp3=temp1+temp2;
                nodes[k].d[1]=sqrt(temp3);
                nodes[k].l[1]=nodes[k].d[1];
                temp1=pow((nodes[k].x1-clust[2].x2),2);
                temp2=pow((nodes[k].y1-clust[2].y2),2);
                temp3=temp1+temp2;
                nodes[k].d[2]=sqrt(temp3);
                nodes[k].l[2]=nodes[k].d[2];
                temp1=pow((nodes[k].x1-clust[3].x2),2);
                temp2=pow((nodes[k].y1-clust[3].y2),2);
                temp3=temp1+temp2;
                nodes[k].d[3]=sqrt(temp3);
                nodes[k].l[3]=nodes[k].d[3];
                temp1=pow((nodes[k].x1-clust[4].x2),2);
                temp2=pow((nodes[k].y1-clust[4].y2),2);
                temp3=temp1+temp2;
                nodes[k].d[4]=sqrt(temp3);
                nodes[k].l[4]=nodes[k].d[4];
                temp1=pow((nodes[k].x1-clust[5].x2),2);
                temp2=pow((nodes[k].y1-clust[5].y2),2);
                temp3=temp1+temp2;
                nodes[k].d[5]=sqrt(temp3);
                nodes[k].l[5]=nodes[k].d[5];
            }
        }
        int temp4=0;
        for(int v=0;v<No;v++)
        {
          if(nodes[v].is_head==0)
          {
           for(int b=0;b<Co;b++)
           {
            for(int m=b+1;m<Co;m++)
            {
             if((nodes[v].d[b])>(nodes[v].d[m]))
             {
                 temp4=nodes[v].d[b];
                 nodes[v].d[b]=nodes[v].d[m];
                 nodes[v].d[m]=temp4;
             }
            }
           }
```

```
  }
 }
//sorted dist for a particular node
for(k=0;k<No;k++)
{
 if(nodes[k].is_head==0)
 {
  for(int m=0;m<Co;m++)
  {
   if((nodes[k].l[m])==(nodes[k].d[0]))
   {
    setcolor(25+m);
    circle(nodes[k].x1,nodes[k].y1,2);
    nodes[k].node_color=25+m;
   }
  }
 }
}
//reelection();
getch();
for(k=0;k<No;k++)
{
 if(nodes[k].is_head==0)
 {
  temp_drain[k]=nodes[k].residual;
 }
}
for(k=0;k<Co;k++)
{
 temp_ech[k]=clust[k].residual;
}
}
no_of_rounds=count_round_head/count_tot_life;
printf("total life  %d %d %d",count_round_head,count_tot_life,no_of_rounds);
getch();
}
//clust_to_white
void clust_to_white()
{
 setcolor(0);
 for(int k=0;k<Co;k++)
 {
  setfillstyle(1,0);
  fillellipse(clust[k].x2,clust[k].y2,5,5);
 }
}
```

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<complex.h>
#include<math.h>
#include<stdlib.h>
#include<graphics.h>

#define No 11
#define am1 11
#define am2 11

struct node
{
int node_id;
int x1,y1;
int is_leaf;
int is_traverse;
int is_col;
}nodes[No];

char temp;
void main()
{
int gd=DETECT,gm=0;//graphics mode changed from text mode
initgraph(&gd,&gm,"c:\\tc\\bgi");
randomize();
int
adm[am1][am2]={{0,0,1,0,0,0,0,0,0,0,0},{0,0,1,0,0,0,0,0,0,0,0},{0,0,0,0,0,0,1,
0,0,0,0},{0,0,1,0,0,0,0,0,0,0,0},{0,0,0,1,0,0,0,0,0,0,0},{0,0,0,1,0,0,0,0,0,0,0},{
0,0,0,0,0,0,0,0,0,0,0},{0,0,0,0,0,0,1,0,0,0,0},{0,0,0,0,0,0,0,1,0,0,0},{0,0,0,0,0,0
,0,0,1,0,0},{0,0,0,0,0,0,0,0,1,0,0}};
int adm1[am1][am2];
char alpa[am1][am2]={"a","b","c","d","e","f","g","h","i","j","k"};
float thres=0.9;
float delta=1.7;
int x=0,y=0,i=0,j=0,count=0;
float
deltamat[am1][am2]={{0,0,0.8691,0,0,0,0,0,0,0,0},{0,0,0.8289,0,0,0,0,0,0,0,0}
,{0,0,0,0,0,0,0.8006,0,0,0,0},{0,0,0.8204,0,0,0,0,0,0,0,0},{0,0,0,1,0,0,0,0,0,0,0
},{0,0,0,0.0273,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0.9611,0,0,0,
0},{0,0,0,0,0,0,0,0.8049,0,0,0},{0,0,0,0,0,0,0,0,0.7532,0,0},{0,0,0,0,0,0,0,0,0.3
759,0,0}};
float feature[am1][am2];
float height[No];
float temp=0;
for(i=0;i<No;i++)
{
height[i]=0;
nodes[i].is_col=0;
}
for(i=0;i<No;i++)
{
nodes[i].is_leaf=0;
nodes[i].is_traverse=0;
}
for(i=0;i<No;i++)
{
for(j=0;j<No;j++)
{
adm1[i][j]=0;
}
}
//line(nodes[1].x1,nodes[1].y1,nodes[2].x1,nodes[2].y1);

//CREATE THE NODES

for(i=0;i<No;i++)
{
x=random(500);
y=random(400);
nodes[i].x1=x;
nodes[i].y1=y;
circle(x,y,7);
nodes[i].node_id=i;
}

//NODE LABEL
for(i=0;i<No;i++)
{
outtextxy(nodes[i].x1-3,nodes[i].y1-2,alpa[i]);
}


//DRAW LINE BETWEEN N0DES BASED ON ADJ MATRIX
for(i=0;i<No;i++)
{
for(j=0;j<No;j++)
{
if(adm[i][j]==1)
line(nodes[i].x1,nodes[i].y1,nodes[j].x1,nodes[j].y1);
```

```
}
}

getch();
cleardevice();

//COPY DELTA MAT INTO TEMP VAR
for(i=0;i<No;i++)
{
for(j=0;j<No;j++)
{
feature[i][j]=deltamat[i][j];
}
}
//CREATE ADJ MATRIX FOR E-GRAPH
for(i=0;i<No;i++)
{

for(int j=0;j<No;j++)
{
if(deltamat[i][j]>=thres)
{
adm[i][j]=0;
}
}
}
cleardevice();

//REDRAW THE NODES
for(i=0;i<No;i++)
{
x=nodes[i].x1;
y=nodes[i].y1;
circle(x,y,7);
}
for(i=0;i<No;i++)
{
outtextxy(nodes[i].x1-3,nodes[i].y1-2,alpa[i]);
}

for(i=0;i<No;i++)
{
for(int j=0;j<No;j++)
{
if(adm[i][j]==1)
line(nodes[i].x1,nodes[i].y1,nodes[j].x1,nodes[j].y1);
```

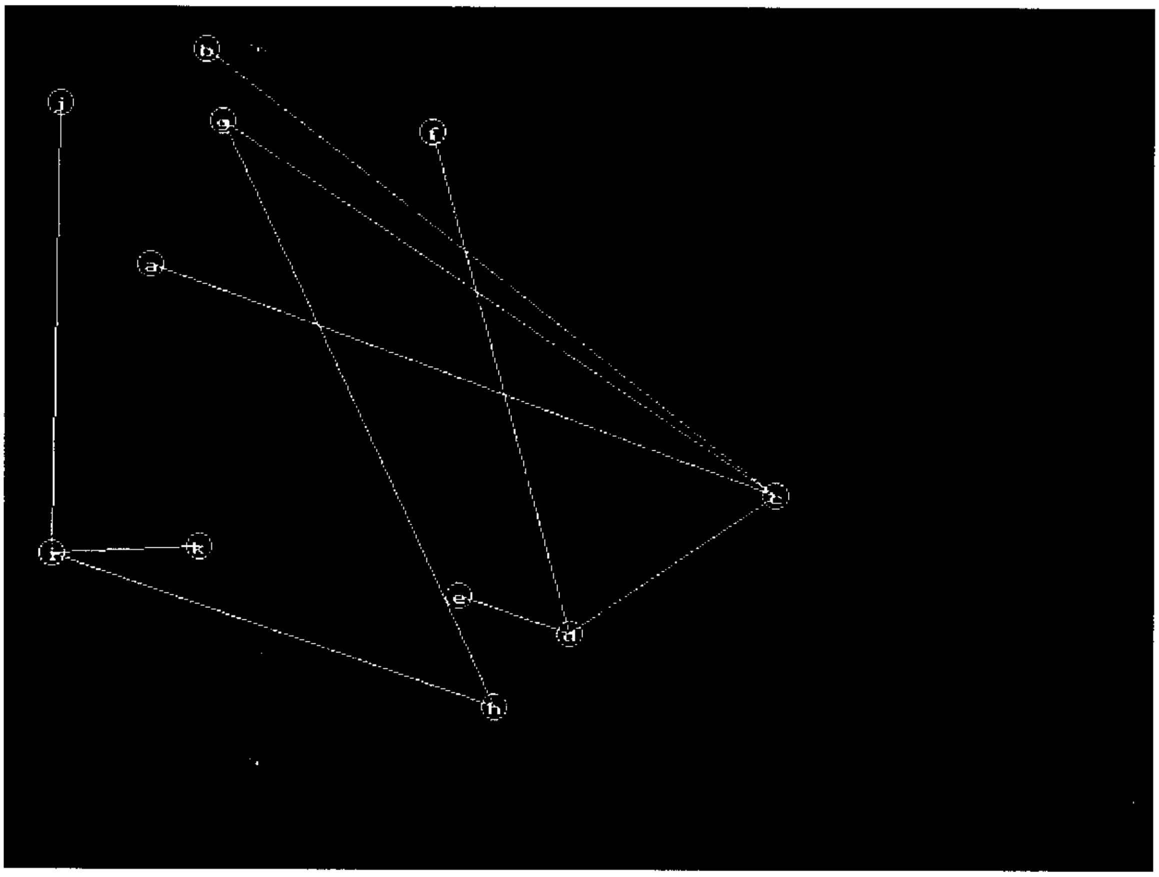
```
}
}

getch();

//FIND LEAF NODES
for(i=0;i<No;i++)
{
for(j=0;j<No;j++)
{
  if(adm[j][i]==0)
  {
    count++;
  }
}
if(count==No)
{
nodes[i].is_leaf=1;
}
count=0;
}

//PASS HEIGHT FROM LEAF TO PARENT
for(i=0;i<No;i++)
{
if(nodes[i].is_leaf==1)
{


for(j=0;j<No;j++)
{

 if(adm[i][j]==1)
 {
 nodes[i].is_traverse=1;

 temp=height[j]+feature[i][j];

 if(temp>=delta)
 {
 setcolor(0);
 line(nodes[i].x1,nodes[i].y1,nodes[j].x1,nodes[j].y1);
 adm[i][j]=0;
 }
 else
 {
```

```
height[j]=height[j]+feature[i][j];
}
}
}
}
}
temp=0;

//PASS HEIGHT OF NON TRAVERSED NODES
for(i=0;i<No;i++)
{
if(nodes[i].is_traverse==0)
{
for(j=0;j<No;j++)
{

 if(adm[i][j]==1)
 {
 nodes[i].is_traverse=1;
 if(height[i]==0)
 {
 temp=height[j]+feature[i][j];
 }
 else
 {
 temp=feature[i][j]+height[i];
 }
 if(temp>=delta)
 {
 setcolor(0);
 line(nodes[i].x1,nodes[i].y1,nodes[j].x1,nodes[j].y1);
 adm[i][j]=0;
 }

 else
 {
 if(height[i]==0)
 {
 height[j]=height[j]+feature[i][j];
 }
 else
 {
 height[j]=feature[i][j]+height[i];
 }
 }
 }
```

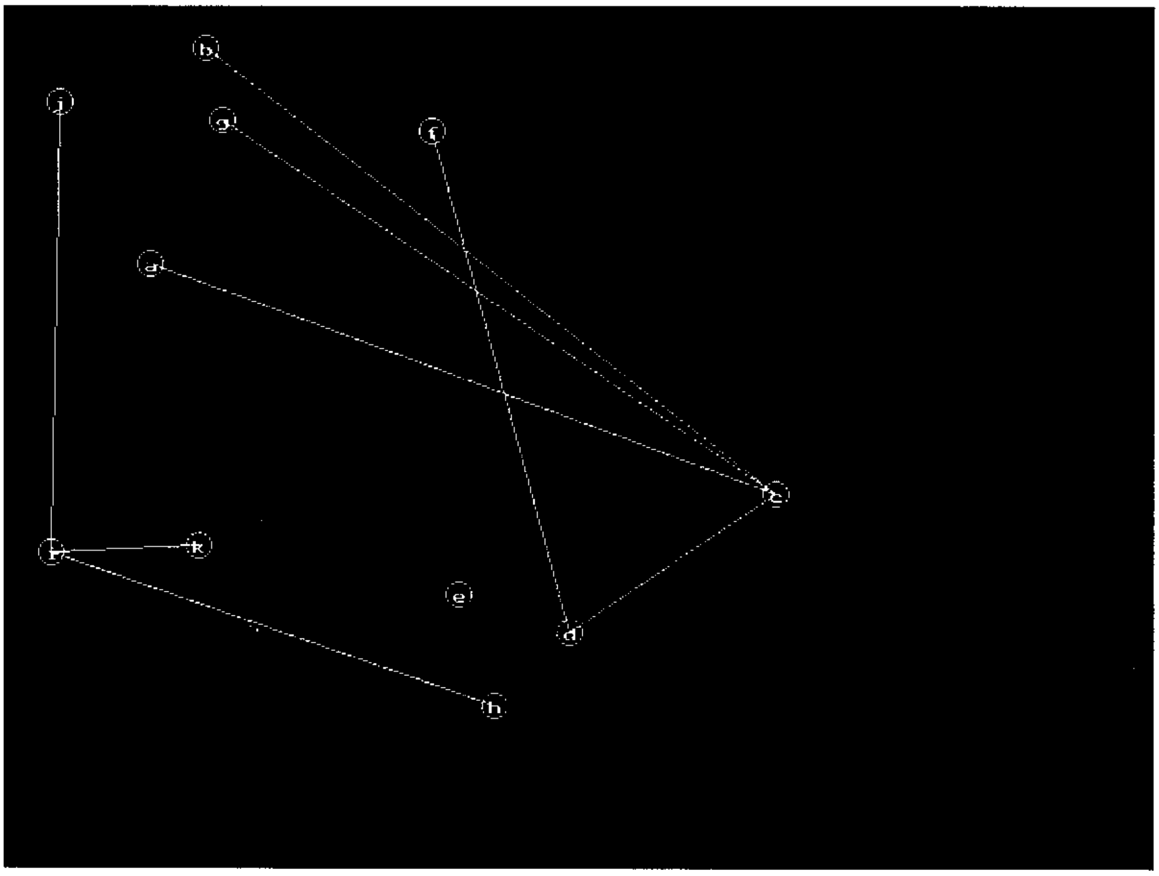
```
}
}
}

for(i=0;i<No;i++)
{
for(j=0;j<No;j++)
{
adm1[i][j]=adm[i][j];
}
}

//SET COLOR TO CLUSTERS

int color=2;
int a=0;
for(j=0;j<No;j++)
{
for(i=0;i<No;i++)
{
if(adm1[i][j]==1)
{
adm1[i][j]=0;
setcolor(color);
circle(nodes[i].x1,nodes[i].y1,7);
circle(nodes[j].x1,nodes[j].y1,7);
line(nodes[i].x1,nodes[i].y1,nodes[j].x1,nodes[j].y1);
for(a=0;a<No;a++)
{
if(adm1[a][i]==1)
{
adm1[a][i]=0;
setcolor(color);
circle(nodes[i].x1,nodes[i].y1,7);
circle(nodes[a].x1,nodes[a].y1,7);
line(nodes[i].x1,nodes[i].y1,nodes[a].x1,nodes[a].y1);
}
}
}
}
color++;
}
getch();
}
```

8.2 Sample Output

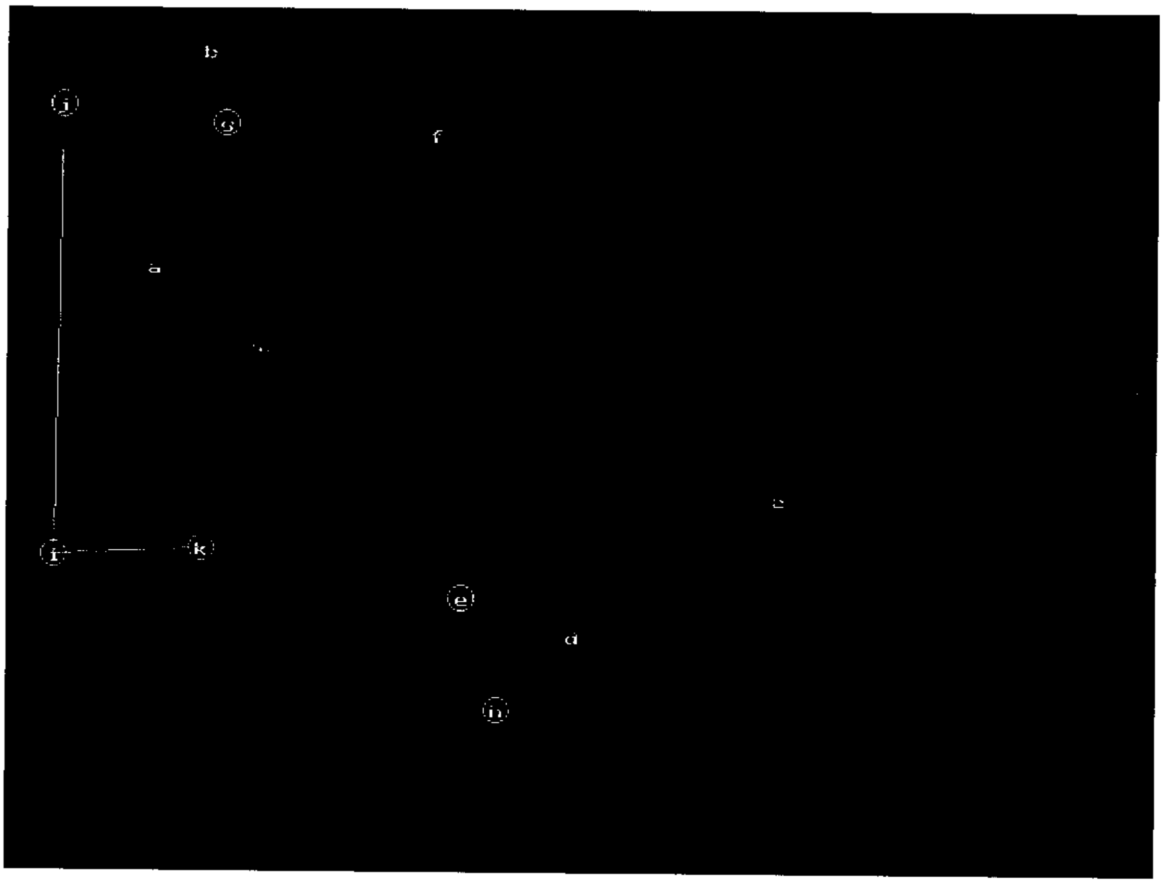
Initial Cluster



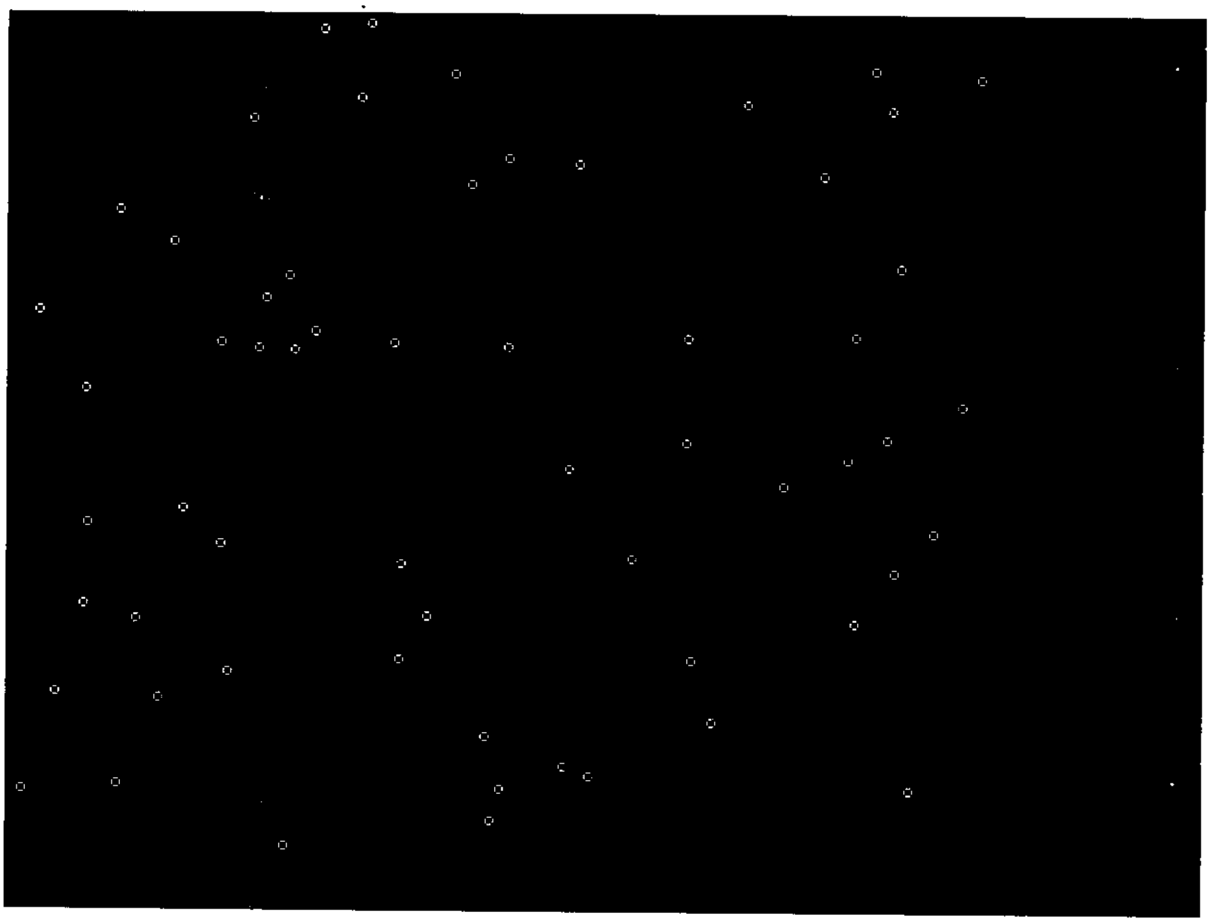
Epsilon Deleted Cluster



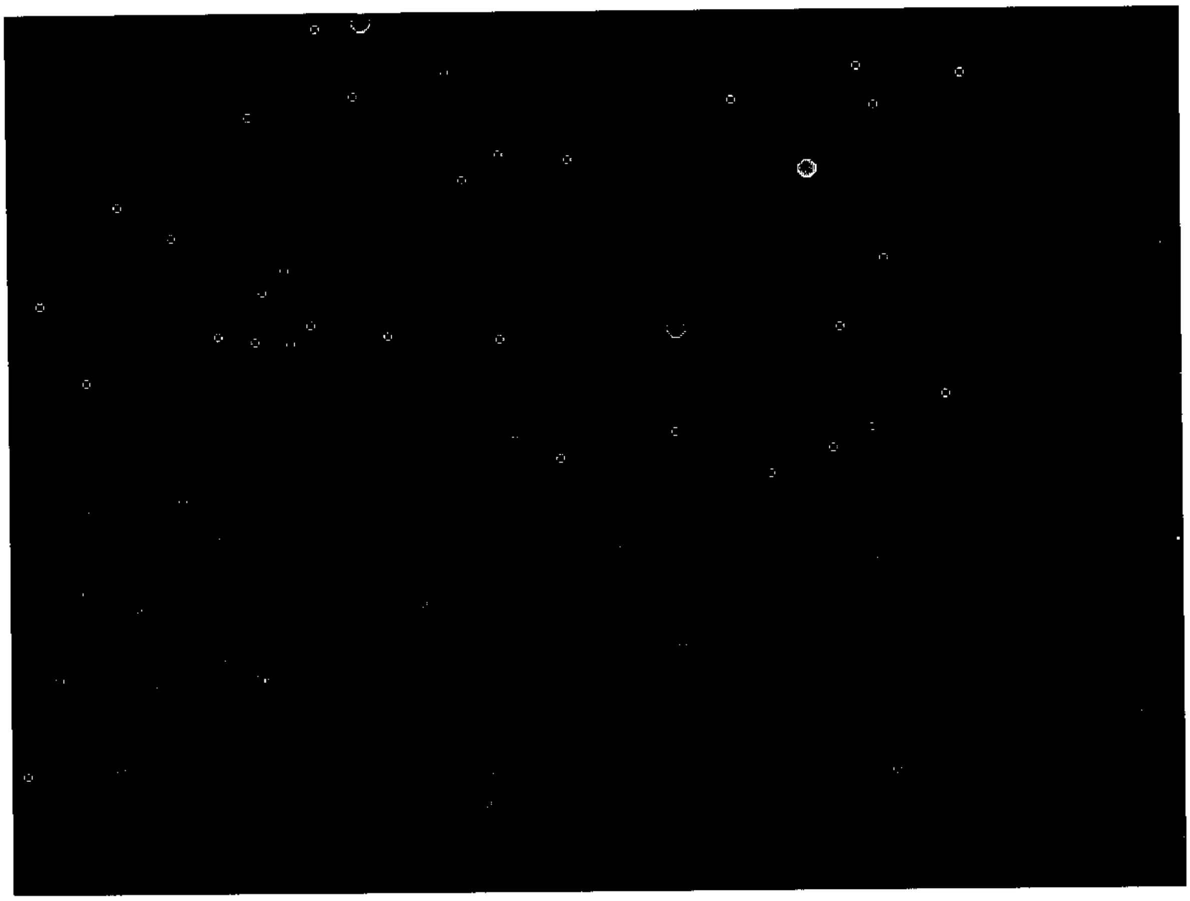
Final Cluster



Randomly Deployed Nodes



Clustered Nodes



9. REFERENCES:

1. Ossama Younis and Sonia Fahmy, "**Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach**," INFOCOMM 2004

2. Anand Meka and Ambuj K. Singh, "**Distributed Spatial Clustering in Sensor Networks.**", EDBT, February 2006

3. Seema Bandyopadhyay and Edward J. Coyle, "**An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks**", IEEE INFOCOM 2003

4.Vivek Mhatre, Catherine Rosenberg, "**Homogeneous vs Heterogeneous Clustered Sensor Networks: A Comparative Study**", IEEE Communications Society 2004

5. Ian F. Akylidy, Shankarasubramaniam, "**A survey on sensor networks**", IEEE Communications Magazine,2002

6.www.pmel.noaa.gov/tao/index.shtml

7. www.epic.noaa.gov/tao/select/timeselect.html

8. www.nationalacademies.org/opus/elnino.html