THESIS

VIRTUAL COORDINATE BASED TECHNIQUES FOR WIRELESS SENSOR NETWORKS: A SIMULATION TOOL AND LOCALIZATION & PLANARIZATION ALGORITHMS

Submitted By

Pritam Shah

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2013

ABSTRACT

VIRTUAL COORDINATE BASED TECHNIQUES FOR WIRELESS SENSOR NETWORKS: A SIMULATION TOOL AND LOCALIZATION & PLANARIZATION ALGORITHMS

Wireless sensor Networks (WSNs) are deployments of smart sensor devices for monitoring environmental or physical phenomena. These sensors have the ability to communicate with other sensors within communication range or with a base station. Each sensor, at a minimum, comprises of sensing, processing, transmission, and power units. This thesis focuses on virtual coordinate based techniques in WSNs. Virtual Coordinates (VCs) characterize each node in a network with the minimum hop distances to a set of anchor nodes, as its coordinates. It provides a compelling alternative to some of the localization applications such as routing.

Building a WSN testbed is often infeasible and costly. Running real experiments on WSNs testbeds is time consuming, difficult and sometimes not feasible given the scope and size of applications. Simulation is, therefore, the most common approach for developing and testing new protocols and techniques for sensor networks. Though many general and wireless sensor network specific simulation tools are available, no available tool currently provides an intuitive interface or a tool for virtual coordinate based simulations. A simulator called VCSIM is presented which focuses specifically on Virtual Coordinate Space (VCS) in WSNs. With this simulator, a user can easily create WSNs networks of different sizes, shapes, and distributions. Its graphical user interface (GUI) facilitates placement of anchors and generation of VCs.

Localization in WSNs is important for several reasons including identification and correlation of gathered data, node addressing, evaluation of nodes' density and coverage, geographic routing, object tracking, and other geographic algorithms. But due to many constraints, such as limited battery power, processing capabilities, hardware costs, and measurement errors, localization still remains a hard problem in WSNs. In certain applications, such as security sensors for intrusion detection, agriculture, land monitoring, and fire alarm sensors in a building, the sensor nodes are always deployed in an orderly fashion, in contrast to random deployments. In this thesis, a novel transformation is presented to obtain position of nodes from VCs in rectangular, hexagonal and triangular grid topologies. It is shown that with certain specific anchor placements, a location of a node can be accurately approximated, if the length of a shortest path in given topology between a node and anchors is equal to length of a shortest path in full topology (i.e. a topology without any voids) between the same node and anchors. These positions are obtained without the need of any extra localization hardware. The results show that more than 90% nodes were able to identify their position in randomly deployed networks of 80% and 85% node density. These positions can then be used for deterministic routing which seems to have better avg. path length compared to geographic routing scheme called “Greedy Perimeter Stateless Routing (GPSR)”.

In many real world applications, manual deployment is not possible in exact regular rectangular, triangular or hexagonal grids. Due to placement constraint, nodes are often placed with some deviation from ideal grid positions. Because of placement tolerance and due to non-isotropic radio patterns nodes may communicate with more or less number of neighbors than needed and may form cross-links causing non-planar topologies. Extracting planar graph from network topologies is known as network planarization. Network planarization has been an important technique in numerous sensor network protocols—such as GPSR for efficient routing, topology discovery, localization and data-centric storage. Most of the present planarization algorithms are based on location information. In this thesis, a novel network planarization algorithm is presented for rectangular, hexagonal and triangular topologies which do not use location information. The results presented in this thesis show that with placement errors of up to 30%, 45%, and 30% in rectangular, triangular and hexagonal topologies respectively we can obtain good planar topologies without the need of location information. It is also shown that with obtained planar topology more nodes acquire unique VCs.

ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support has made the completion of this thesis possible.

First, I would like to express my thanks and gratitude to my advisor Prof. Anura Jayasumana for his valuable guidance and support in achieving my Master's. Thank you for all the help, valuable time and giving me an opportunity to work with you. A sincere thank you to other committee members: Prof. Sudeep Pasricha and Prof. Yashawant Malaiya. Thank you for serving as a committee member and providing your valuable time when needed.

I am grateful to all the colleagues at CNRL lab. Special thanks to Dilum Bandara, Vidarshana Bandara, Sahil Mehta for sharing their enthusiasm and criticism on my work. A special thanks to Dulhanjalie Dhanapala for her help, suggestions and thorough reviews on my work. I would like to thank all my peers, colleagues at CSU, who either directly or indirectly contributed to completion of this work, for all their support and encouragement.

Last, but never the least, I am indebted by the love, support, and understanding offered by my family members. I thank my family for their belief and constant encouragement. None of this work would have been possible without your unyielding support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

With recent advances in technology such as sensing materials, processing units, and transceivers sensor nodes have become feasible. The future will see the integration of abundance of existing technology with wireless sensor networks. This chapter gives an overview on context of this thesis in Wireless Sensor Networks (WSNs).

1.1 Introduction to Wireless Sensor Networks

A wireless sensor network consists of spatially distributed sensors that monitor physical or environmental conditions and cooperatively pass their data through the network to other nodes or to main location. Each sensor node is comprises of, at a minimum, a radio transceiver, sensor, processor, and an energy source, usually a battery or an embedded unit for energy harvesting. The size of WSNs could vary from a few to several hundreds or even thousands of sensor nodes. A sensor node may vary in size from that of a shoebox down to the size of a grain of dust, although functioning "motes" of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from a few to hundreds of dollars at the upper end to a few dollars at the lower end, depending on the complexity of the individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and communications bandwidth. The nodes communicate wirelessly and often self-organize after being deployed in an ad-hoc manner. Routing algorithms and deployment of sensors play a crucial role in self-organization of network.

1.1.1 Protocols in WSNs

Since a wireless sensor network is a distributed real-time system a natural question is how many solutions from distributed and real-time systems can be used in these new systems? Unfortunately, very little prior work can be applied and new solutions are necessary in all areas of the system. The main reason is that the set of assumptions underlying previous work has changed dramatically. Most past distributed systems research has assumed that the systems are wired, have unlimited power, are not real-time, have user interfaces such as screens and mice, have a fixed set of resources, treat each node in the system as very important and are location independent. In contrast, for wireless sensor networks, the systems are wireless, have scarce power, are real-time, utilize sensors and actuators as interfaces, have dynamically changing sets of resources, aggregate behavior is important and location is critical. Many wireless sensor networks also utilize minimal capacity devices which places a further strain on the ability to use past solutions.

1.1.2 Applications

Currently, WSNs are beginning to be deployed at an accelerated pace. Applications of WSN include but not limited to environment monitoring (e.g. forest fire detection, landslide detection, water quality monitoring), industrial monitoring (e.g. data logging, machine health monitoring), agriculture (e.g., irrigation systems), traffic control, and home automation. Figure 1.1 shows field applications of WSNs. Figure 1.1 (a) shows urban environment monitoring application. Figure 1.1 (b) shows an application called AggreGate from Tibbo Technologies for green houses. Greenhouse always has to be in a certain temperature and humidity percentage. These variables keep changing at various times of the day, and as equipment and workers enter and leave the warehouse. Sensor grid, deployed throughout the warehouse, could connect to

AggreGate server and constantly relay sensor data. When a certain threshold is crossed (i.e., humidity too low) an alert could be raised and corrective action will automatically occur - water atomizers would kick in, adding humidity. User could generate reports to easily see how much power and water greenhouse consumes, and uses the data to plan the space layout more efficiently and minimize any waste of resources.

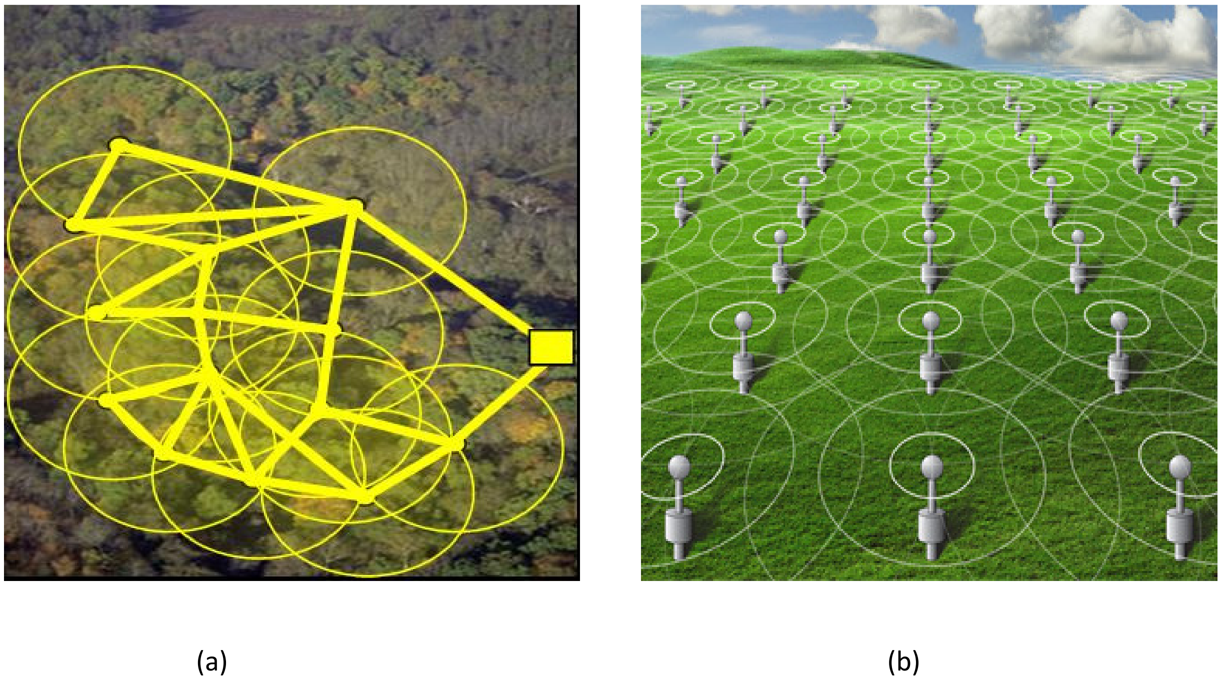


Figure 1.1 Applications of WSNs in field monitoring: (a) Environment monitoring [DoKo11], and (b) Sensor grid deployed throughout the warehouse [TiTe11].

1.1.3 Virtual Coordinate Space

Internet and MANET routing techniques do not perform well in WSN. Routing in WSN is broadly divided into two categories: Content based and address based. Content based routing protocols use information within the packet while address based routing protocols use nodes' address to forward packet to destination. Content based protocols need flooding for route discovery. Because of many redundant message transmissions flooding is not efficient in large sensor networks. As an answer to this problem, address based routing protocols were investigated in the context of WSNs. The main drawback of this approach is that nodes need to know their location, which is demanding in terms of energy and/or hardware costs. Virtual coordinates (VCs) were introduced to overcome this problem. Here, a node infers its locations from communicating with a small subset of location-aware anchor nodes. Such location-aware anchor nodes need to be positioned manually. A more general solution is to have location-unaware anchor nodes elected by the network. This thesis concentrates more on Virtual Coordinate Space (VCS) for wireless sensor networks.

1.2 Motivation and Problem Statement

This section of thesis presents three identified problems in WSNs.

1) Recent advances in technologies have made wireless sensor networks feasible. But it is still costly to build WSN nodes only for experiments, as a test node involves many other debugging tools or sometimes only due to sheer number of test nodes required. Running real experiments on WSNs testbeds is time consuming, costly, difficult and sometimes not feasible given the scope and size of application. Besides, repeatability of experiment under same conditions is largely comprised since many factors affect the experimental results at the same time such as packet loss, radio interference, clock synchronization etc. For example, it is possible that due to temperature changes or any physical motions experimental results changes in real world. Simulation is, therefore, the most common approach for developing, and testing new protocols and schemes for sensor networks. There are number of advantages of using simulator including lower cost, ease of implementation, repeatability, and practicability of testing larger-scale networks. This has led to many simulators for wireless sensor networks.

Many commercial and open source simulation tools like ns-2, Omnet++, Opnet, TOSSIM are available for networking research. It is important to know the features offered by different simulators to choose an appropriate one depending on the application and accuracy required. No single simulator is suitable for simulating all types of applications on any scale. Sufficient amount of survey is available on comparison of WSN simulators [JeZo09, Yu11, SuLi11, and EgVa05]. This thesis concentrates more on virtual space for WSNs. Therefore, though many simulators are available none satisfied our requirement of simple interface for virtual coordinates. The unavailability of such a simulation tool has important implications. Wireless sensor network is defined by lot of parameters. Each node has its characteristics. There are lots of factors involved in simulating WSNs such as Network Size, Node Deployment, Communication range, Node energy etc. [RoMa04]. Network size depends upon the area to be sensed or monitored. In the area of interest nodes could be deployed randomly (as dropped from airplane) or manually. They could either be sparsely deployed or densely deployed depending on the sensing range of each node, accuracy required, communication range or availability of nodes. Having no tool consumes a lot of precious time constructing different topologies, and implementing well-known algorithms for comparison. It is important to test newly developed algorithm on different types of WSNs before coming to conclusion. Creating topologies with different parameters by writing code from scratch is time consuming even in high level language like MATLAB. Also, more research in the area is not encouraged if building basic framework for experiments consumes lot of time.

2) Another problem this thesis identifies is node localization and routing in regular deployed networks. Wireless sensor network deployments can be either deterministic or randomized [AlKa04]. In deterministic deployment, the sensors are manually placed at deliberately chosen

spots and data travels through pre-determined paths [AlKa04]. Out of all the 15 major applications of WSNs studied in [RoMa04], in 13 cases nodes were deployed manually. Node localization is the problem of determining the geographical location of each node in the system. Localization is one of the most fundamental and difficult problems that must be solved for WSN. Localization is a function of many parameters and requirements potentially making it very complex. For example, issues to consider include: the cost of extra localization hardware, do beacons (nodes which know their locations) exist and if so, how many and what are their communication ranges, what degree of location accuracy is required, is the system indoors/outdoors, is there line of sight among the nodes, is it a 2D or 3D localization problem, what is the energy budget (number of messages), how long should it take to localize, are clocks synchronized, does the system reside in hostile or friendly territory, what error assumptions are being made, and is the system subject to security attacks? For some combination of requirements and issues the problem is easily solved. If cost and form factor are not major concerns and accuracy of a few meters is acceptable, then for outdoor systems, equipping each node with GPS is a simple answer. Most other solutions for localization in WSN are either range-based or range-free. Range-based schemes use various techniques to first determine distances between nodes (range) and then compute location using geometric principles. To determine distances, extra hardware is usually employed, e.g., hardware to detect the time difference of arrival of sound and radio waves. This difference can then be converted to a distance measurement. In range-free schemes distances are not determined directly, but hop counts are used. Range-free solutions are not as accurate as range-based solutions and often require more messages. However, they do not require extra hardware on every node. This thesis attempts to find range-free localization from

VCs in regular networks. Once the node locations are known it is possible to use the underlying known topology of regular networks to find the multiple paths more efficiently.

In WSNs usually after obtaining the locations; many routing algorithms use greedy forwarding, where the packet is forwarded to the neighbor nearest to the destination neighbor. Though efficient, it only finds single path for same source destination pair. Multi-path routing is proved better for various reasons such as better throughput, network efficiency. Other reason being single and so lower energy paths may not be optimal from the point of view of network lifetime and long-term connectivity [ShRa02]. Because only if nodes in center of the network continues to provide these paths they will run out of energy soon and it will form a network discontinuation. To overcome this disadvantage, [ShRa02] proposed an idea to find multiple paths between source and destinations, and one is then chosen based on the energy metric. By simulations they show this technique provides network connectivity for longer time. It uses localized flooding to find all the routes from particular source to destination. Though, flooding is useful because it guarantees finding the multiple paths from source to destination, at the same time it also propagates many unnecessary messages all over the network exhausting the network resources, reducing the network lifetime. In Chapter 4 it is shown that if a location of node is known in regular networks instead of greedy forwarding, one can use vectors to define path and improve path length.

3) The third and final problem this thesis identifies is in topology planarization. The sensor nodes are often battery powered and therefore limited in power. All the schemes used for localization, routing, processing in WSNs should be as energy efficient as possible. Topology control is important in WSN because even it can help save energy. As mentioned in [HuHs08, SaWe01], pre-configured WSNs are superior in network performance for many aspects,

including end-to-end delay time, data delivery ratio, and energy consumption, compared to random topologies. Many manually deployed networks are positioned in the form of hexagonal, rectangular or triangular grids. In rectangular grid each node is connected to four nearest neighbors, in triangular grid each node is connected to its six nearest neighbors, and in hexagonal grid each node is connected to its three nearest neighbors. But in real world, manual deployment is not possible in exact rectangular, triangular or hexagonal grid. Due to placement constraints, nodes are often placed with some placement tolerance. Because of placement tolerance and due to anisotropic radio patterns node can communicate with more number of neighbors than needed. The topology gets deformed due to cross connections or more number of neighbors. All the network parameters like routing, energy consumption degrades due to unknown topology, excess of interference and collision. The third problem this thesis addresses is to obtain regular topologies from deformed topologies caused due to placement errors and asymmetric radio patterns.

1.3 Contributions

This thesis has three main contributions.

1) First contribution of this thesis is VCSIM – a MATLAB based wireless sensor network simulator with simulation options for virtual coordinates. This thesis describes features and application of this tool in detail in Chapter 3. With the help of this tool one can easily create or import a network with different parameters such as size, shape, and communication range. Graphical User Interface (GUI) provided with VCSIM offers visualization of WSN. It is possible to change network size, node density, node distribution without much effort. If user wishes to, it is possible to individually place each node in a network. Virtual coordinates are obtained by communicating with small subset of nodes called anchors. VCs of nodes in a same network

change with different anchor placement affecting routing performance, localization, topology maps or any other VCs based techniques. Therefore, overall anchors play an important role in assigning VCs. GUI of VCSIM provides a simple interface for anchor placement with three options. With this tool one can either choose anchor nodes randomly or use an anchor selection algorithm such as Extreme Node Search (ENS) [DhJa11a]. VCSIM provides an option to export virtual coordinates or connectivity matrix of nodes for user needs. It can generate topology preserving maps using three algorithms namely multi-dimensional scaling (MDS), Singular value decomposition (SVD) and Directional Virtual Coordinates (DVC).

2) Second contribution of this thesis is on localization in regular networks. Chapter 4 of this thesis presents a transformation to find location of nodes in rectangular, triangular and hexagonal networks from VCs. These locations are obtained without any special hardware. Once locations of nodes in regular networks are found a multipath routing algorithm is presented.

3) The final contribution of this thesis is in topology control. In certain applications, such as security sensors for intrusion detection, agriculture or land monitoring, or for fire alarm sensors in a building, the sensor nodes are deployed in advance in an orderly fashion [HuHs08]. In real world, manual deployment is not done in exact regular rectangular, triangular or hexagonal grid due to placement constraints. Because of placement tolerance and due to anisotropic radio patterns a node can communicate with more or less number of neighbors than needed. The topology gets deformed due to cross connections or more number of neighbors. In Chapter 5 of this thesis an algorithm is presented to remove unnecessary links to form an exact rectangular, triangular or a hexagonal topologies. This would help in obtaining the known topologies, and therefore in efficient routing schemes and improving the network performance.

1.4 Outline

Rest of the thesis is organized as follows. Chapter 2 discusses background information related to virtual coordinate systems, directional virtual coordinates, routing, and topology control in wireless sensor networks. It also reviews features and limitations of available simulation tools for WSNs. Chapter 3 describe the VCSIM and usefulness of application. It lists all currently available features of VCSIM in detail. A method for localization and routing in regular networks is presented in Chapter 4. Chapter 5 presents the topology planarization algorithm for deformed rectangular, triangular, and hexagonal topologies. Chapter 6 concludes the thesis with summary and future work. The source code of the VCSIM tool is included in the appendix.

Chapter 2

Background Information

This chapter gives the background information related to virtual coordinate systems and directional virtual coordinates. Directional virtual coordinates are used in Chapter 3 to identify direction of neighbors. This chapter also explains the routing techniques used in WSNs. It is important that reader is aware of these concepts, to understand the thesis in its full context. This chapter also discusses importance of topology control (TC) and different techniques for TC in WSNs.

2.1 Routing in WSNs

Standard Internet routing achieves scalability through address aggregation, in which each route announcement describes route information for many nodes simultaneously. Routing in WSNs is challenging due to one fact that these networks are ad hoc, and therefore it does not rely on a preexisting infrastructure, such as routers in wired networks or access points in managed (infrastructure) wireless networks. Instead, each node participates in routing by forwarding data for other nodes, and so the determination of which nodes forward data is made dynamically based on the network connectivity. Currently routing protocols for sensor networks typically fall into two categories: content-based and address-based. Content based addressing scheme uses information available in the packet to define the destination set while address-based uses some sort of position information [CaAb04]. In certain applications content-based algorithms are used in an initial setup stage to discover source and/or destination information, but then it switches to an address based protocols for subsequent transfers. Content based protocols, are query based [AkYo05], and usually rely on flooding techniques e.g. rumor routing [BrEs02], Directed

Diffusion [InGo03], SPIN [HeKu99]. Therefore, they consume more bandwidth compared to address-based routing techniques [CaAb04].

Address-based routing uses nodes' either geographic or logical position information to relay the data to the desired region rather than the whole network. The most widely known geographic location based proposal is Greedy Perimeter Stateless routing (GPSR) [KaKu00], it forwards packets (when possible) in a greedy manner towards the destination. Several other address-based schemes have been proposed e.g. Geographic and Energy Aware Routing (GEAR) [YuEs01], Greedy Other Adaptive Face Routing (GOAFR) [KuWa03a]. These routing schemes are based on the assumption that geographic information for each node is available either through GPS or some other localization means. Though, a plenty of localization algorithms have been proposed based on different techniques like received signal strength measurement (RSSI), Angle-of-Arrival (AoA), GPS receivers, triangulation etc., it still remains a hard problem [CaAb04]. Equipping nodes with GPS is costly and infeasible in many applications due to energy limitations. In practice, RSSI ranging measurements contain noise on the order of several meters. This noise occurs because radio propagation tends to be highly non-uniform in real environments [BaTa05]. It has been found that errors in node positions may lead to unrecoverable routing failures [SeHe04]. Further, geographic routing also suffers from the dead-end problems, also known as local minima problems, due to physical voids. Local minima problem is where greedy routing fails because a node has no neighbor closer to the destination than the node currently holding the packet.

2.2 Virtual Coordinate System (VCS) and Directional Virtual Coordinates (DVC)

Virtual Coordinate system (VCS) provides a compelling alternative for structuring WSNs without location information. In virtual coordinate systems each node is characterized by a VC vector which consists of minimum hop distances to each of the anchors. Virtual coordinates do not give the accurate representation of the underlying geography but, characterizes nodes using connectivity based distance. Physical voids are transparent in Virtual Space (VS), but if virtual coordinates are not obtained from properly deployed anchors, the network suffers from identical logical coordinates and local minima problem. Significant amount of research is available on VCS and VCS based routing. There are several approaches proposed for finding Virtual Coordinates [CaAb04], [RaRa03], [LiAb08], [TsYa09], [CaCh05]. In [RaRa03], a number of furthest apart nodes called perimeter nodes are selected as anchors. Authors study a scenario when nodes know neither their location, nor whether they are on perimeter. Each node builds its perimeter vector, i.e. the distances from that node to all perimeter nodes. Each node then computes normalized coordinates for both itself and the perimeter nodes using inter-perimeter distances. All nodes except perimeter nodes then run a relaxation algorithm to find their own virtual coordinates. These generated VCs are then used in greedy forwarding. At local minima, it performs an expanding ring search till a closer node is found or TTL expires. In [CaCh05], authors propose a new method called Virtual Coordinate assignment protocol (VCap). In VCap, virtual coordinates are obtained using three rightly selected anchors. These generated virtual coordinates are then used for greedy forwarding. Though VCap is simple and has less overhead compared to [RaRa03], main disadvantage is of having identical coordinates. As a solution authors define a *zone* of nodes with identical coordinates so the packet is delivered to a zone and then the packet is sent to destination using proactive based approach. [CaAb04] uses the VC vector as it is as obtained from the hop distances to anchors. It uses greedy forwarding whenever possible. In case of local minima backtracking is used where packet is returned back and forwarded to node excluding the nodes from which it came back or any node that packet has visited earlier. Major disadvantage is each node should keep the history in packet header. In [LiAb08] authors claim that a local minimum arises in VCs due to quantization noise. Simply because VC is always an integer number, so they propose a method called Aligned VCS, where nodes calculate its VCs by averaging its own coordinates with neighboring coordinates.

[DhJa09] uses subset of three anchors to bring packet out of local minima. As triangle is always a convex polygon if the source and destination VCs are formed using same three anchors such that both nodes are within a triangle formed by anchors it can use the greedy forwarding. If in-between node has local minima a different set of three anchors is chosen. [DhJa11c] uses a combination of both geographic and logical addresses to avoid local minima in so called Geo-logical routing.

Though VCs are simple to obtain than geographic coordinates they have their own inherent disadvantages. VCs are insensitive to directions. [DhJa11b] identifies this problem and proposes a directional virtual coordinates (DVC). Consider a node i in 2D network. $h_{n_iA_j}$ and $h_{n_iA_k}$are two VCs with respect to anchor A_j andA_k. Then the magnitude of DVC at node i in the direction A_jA_k is given as

$$\boldsymbol{DVC(n_i) = f\left(h_{n_iA_j}, h_{n_iA_k}\right) = \frac{1}{2*h_{A_jA_k}}(h^2_{n_iA_j} - h^2_{n_iA_k})} \qquad (2.\ 1)$$

In [DhJa11a] it is showed that by calculating the difference between DVCs of neighboring nodes, one can get a good approximation of position of node in rectangular network. In Chapter 4 of this thesis a transformation is proposed to calculate position of node in rectangular network by using VCs. This thesis also presents the transformation to calculate positions of nodes in triangular and hexagonal network.

2.3 Simulators for WSNs

Simulation is the most common approach in developing and testing of new schemes in WSNs. WSNs simulators allow users to isolate different factors by tuning configurable parameters. Commercial tools, like Opnet and Qualnet, and open source tools, like ns-2, Omnet++, are full blown network simulators, which include all major communication protocols. Some of them like NS-2, OMNET++, J-Sim are general purpose simulators while TOSSIM, SENS, Prowler are specific WSN frameworks. Different approaches are used for simulation and modeling of SN and WSN. They operate in different levels of simulation: hardware emulation, operating system and application level. The supported hardware platforms are mainly TmoteSky, MICA, MICA2, MICAZ etc. Given the facts that a number of sensor network simulators are available, the most effective tool is chosen based on its suitability to implement the particular model. A detailed survey and comparison of WSN simulation tools can be found in [KoSa09, Yu11, SuLi11, and EgVa05]. Aim of this thesis is to study various schemes on virtual coordinates. It is assumed that these sensors are able to communicate with each other with available operating system and hardware. A simple MAC without packet loss is assumed. This simulator does not try to verify or implement existing communication protocols such as ZigBee, Bluetooth or any physical radio loss. To the best of our knowledge, there is no tool available for study related to virtual coordinates. One of the thesis paper from ETH, Zurich implements virtual coordinate generation method proposed in [RaRa03].

2.4 Topology Planarization in WSN

A topology of a multihop wireless network is the set of communication links between node pairs used explicitly or implicitly by a routing mechanism. A basic requirement for an edge (a, b) is that the nodes a and b are within each other's transmission range. Topology Control (TC) is a technique in which a node can carefully select a set of neighbors to establish logical data links and dynamically adjust transmitting power (i.e. transmission radius) for different links [Hu93].

Topology Control (TC) is one of the most important techniques used in wireless ad hoc and sensor networks to reduce energy consumption (which is essential to extend the network operational time) and radio interference (with a positive effect on the network traffic carrying capacity) [WiLa10, Sa05, HoLi05]. The goal of this technique is to control the topology of the graph representing the communication links between network nodes with the purpose of maintaining some global graph property (e.g., connectivity), while reducing energy consumption and/or interference. Lately, topology control is divided into two sub-problems: topology construction and topology maintenance. Topology construction algorithms help in building the initial reduced topology while topology maintenance is the process that changes the reduced topology from time-to-time.

Topology construction can be exercised in different ways. The initial topology can be reduced by solving the Critical Transmission Range (CTR) problem, which reduces the transmission range of all nodes by the same minimum amount, or by solving the Range Assignment Problem (RAP), which sets the minimum transmission range for each node. Other topology construction algorithms are based on the Connected Dominating Set (CDS) paradigm [WiLa09]. Here, the idea is not to change the transmission range of the nodes but to turn unnecessary nodes off while preserving important network properties, such as connectivity and communication coverage.

The performance of WSNs is greatly influenced by their topology. A desirable network topology not only reduces energy consumption and prolong network lifetime, but also improves spatial reuse and mitigate the medium-access control (MAC) level contention. Much research has been done on patterned/ regular topologies in WSNs as they can efficiently save energy and achieve long networking lifetime [Ti05, SaWe01, ShQa08, St97, HeXi06, ShDi07, and GuWa07]. Though a significant contribution in both the fields, topology control and regular topologies, nobody has attempted to achieve regular topologies by removing redundant links caused due to placement errors or anisotropic radio patterns.

Chapter 3

VCSIM - A Virtual Coordinate Space Simulator for WSN

As described in Chapter 2, in many cases it is impractical to experiment on real wireless sensor network systems. An example of this is a low power system-on-chip platform that, while possible, is not yet practical due to up-front design, and fabrication costs. Second, even if the hardware platform exists, it may be prohibitively expensive to deploy a large-scale network. For example, new protocols developed in research or applications to be deployed have to be evaluated with respect to factors such as scalability and reliability, which require hundreds or thousands of nodes to evaluate. With current nodes costing approximately one hundred dollars, such evaluations could cost thousands to tens of thousands of dollars. Third, even if it is practical to evaluate research on the real hardware platform, it may not be practical to experiment for different protocols or for different topologies with same parameters. Therefore, simulation is important in study and development of wireless sensor networks.

VCSIM is a wireless network simulator specifically designed for simulating wireless sensor networks and algorithms operating in the virtual coordinate space. This tool facilitates the creation of network with different parameters like range, size, shape, distribution, etc. Simulator provides a simple interface for anchor placement and obtaining VCs. This chapter describes the VCSIM in detail, together with its features and applications. The appendix offers the source code of this tool implemented in MATLAB environment.

3.1 Features

VCSIM offers a variety of options for designing and experimenting with WSN with its graphical user interface. GUI of VCSIM is as shown in Figure 3.1.

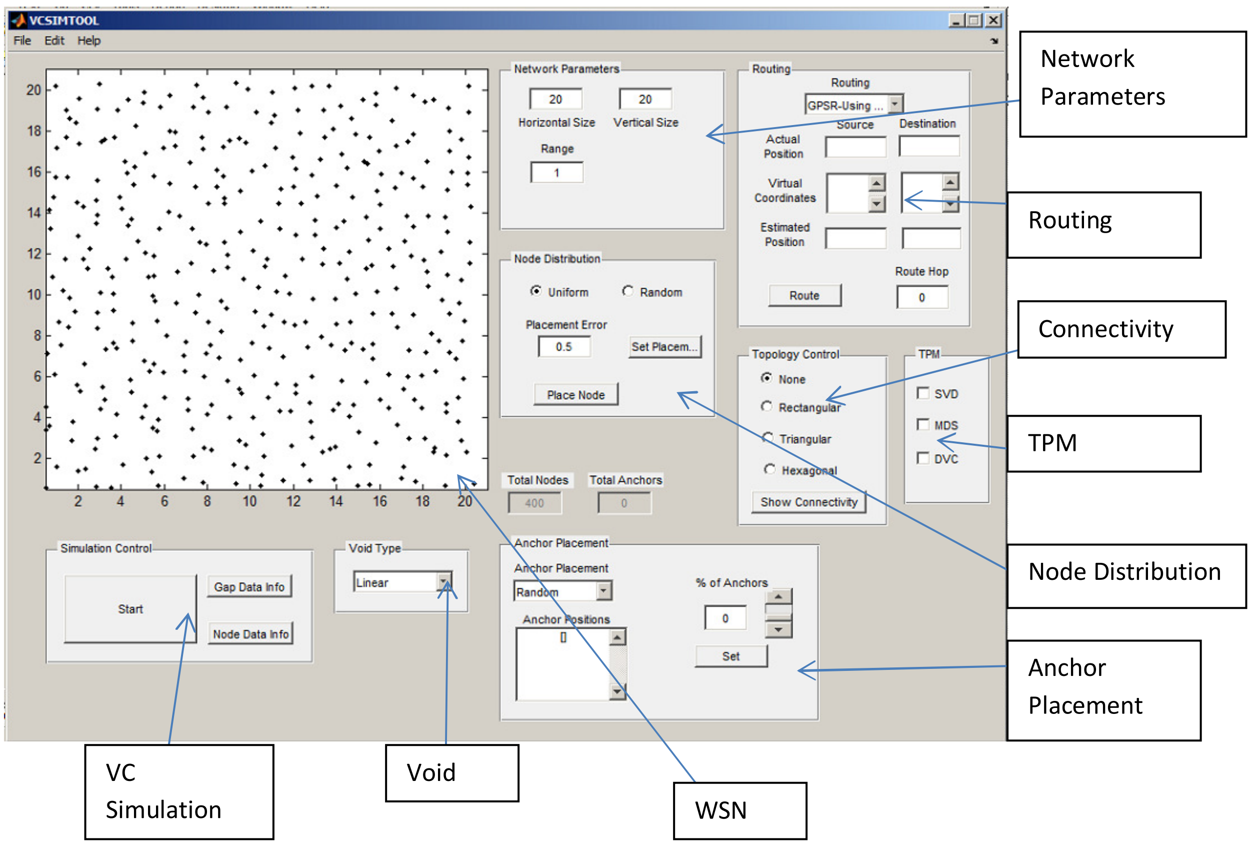


Figure 3.1 GUI of VCSIM

This section provides a brief user guide on how to use the simulation tool and all its available options.

3.1.1 Network Layout Generation

The first step is to have a clear understanding of the simulation scenarios to be run. Here, the user needs to know in advance, what kind of network him or she wants to experiment with. User can define network of any geographical size as long as unit for defining range, network size, and placement tolerance is kept constant. Following parameters help user in creating the network.

(a) **Network Size:** User can easily create the network with VCSIM GUI without the need of writing code for it. User first decides the size of area on which network is deployed. Size is automatically displayed for imported networks. The tool is limited to lowest size of 10×10 unit square area.

(b) **Node Distribution:** Using this menu user can opt to distribute nodes in uniform or random manner in a given size. In uniform distribution nodes are placed one unit distance apart from each other on grid without any obstacles or missing modes. In random deployment user needs to select a node density. Node density is a percent number of sensors in unit square area, placed one unit distance apart. For example, if user needs to deploy 200 sensors in a 400 unit square (20 unit × 20 unit) area then the node density would be 50. User can also import a network from file menu, which reads a file containing either the location coordinates of nodes or as matrix containing presence or absence of node.

(c) **Place Node:** If user wants to place a single node somewhere in network he or she does not have to go through all the procedure again. A single node can be placed using mouse by selecting toggle button option “Place Node”. User can also opt to create a network by placing all nodes one by one using this “Place Node” option.

(d) Move Node: This option is useful in connectivity related information. For example, connectivity matrix can be changed if node moves further away from other nodes. Each node can be moved individually by using this option.

(e) Range: Range defines the communication range of each node. Range is particularly important to get the connectivity between all nodes. To get all network connectivity and topology maps from MDS or SVD node range should be at least equal to maximum distance between two neighboring nodes.

(f) Placement Tolerance: By default the nodes are placed at grid vertices. Placement tolerance is indication of allowed error in placing the node at exact grid vertices. For example, 0.1 = 10% placement tolerance would mean, node is placed anywhere in 0.1 unit radius from its actual location. Therefore, a node with coordinates [1, 1] would be represented as [1±0.1, 1±0.1].

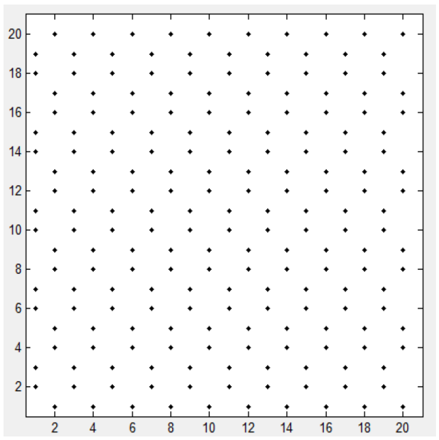
(g) Voids: It is possible that though uniform everywhere, it has some obstacles like a wall, a lake or a street where sensor placement is not possible in real world deployments. In simulator these obstacles are made realizable by using voids. Voids can be created using following four shapes or composites of those shapes:

a) Ellipse: With this option user can create a void of elliptical shape.

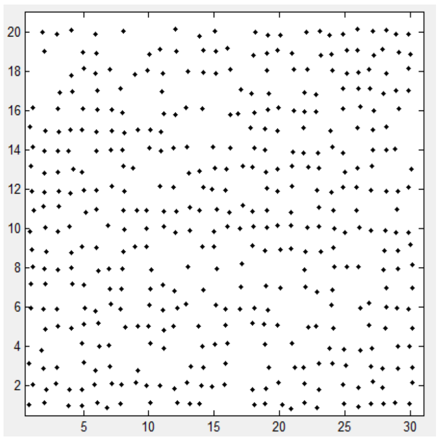
b) Rectangle: With this option user can create rectangular voids.

c) Interpolation: Here user selects certain boundary points on a void. A closed polygon is created using spline interpolation technique.

d) Linear: Here user selects the vertices of polygon. A closed void shape is formed by joining these vertices with straight lines.

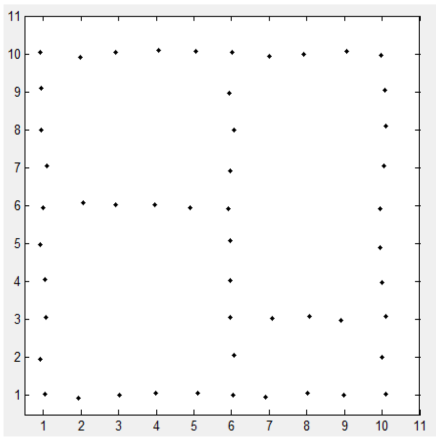


(a)

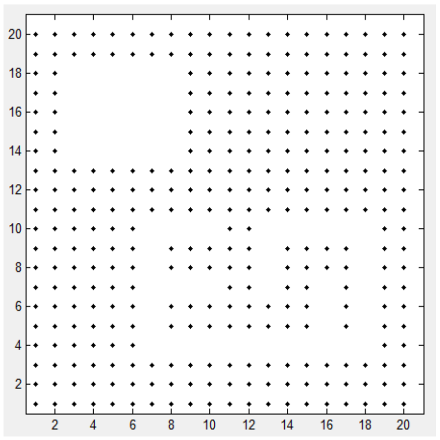


(b)

Figure 3.2 (a) Hexagonal network on a 20×20 grid with 200 nodes, and (b) Random topology on a 30×20 grid with placement tolerance 20% and node density 70%.

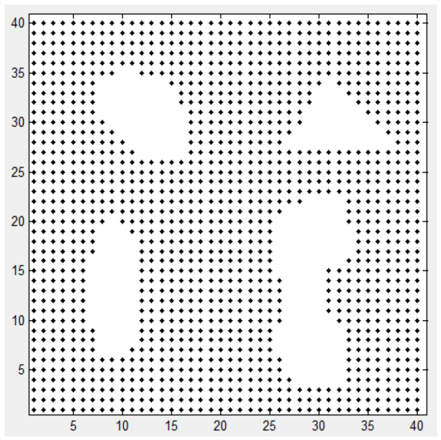


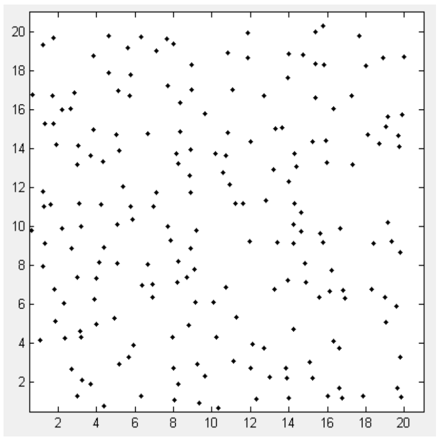
(a)



(b)

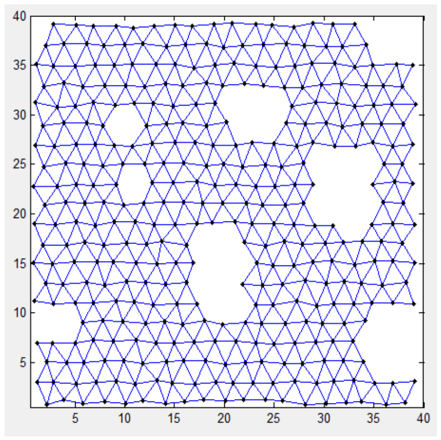
Figure 3.3 (a) Manually generated topology on a 10×10 grid with 10% tolerance, E.g. Buildings, Street lights, and (b) Manually generated topology with obstacles on a 20×20 grid

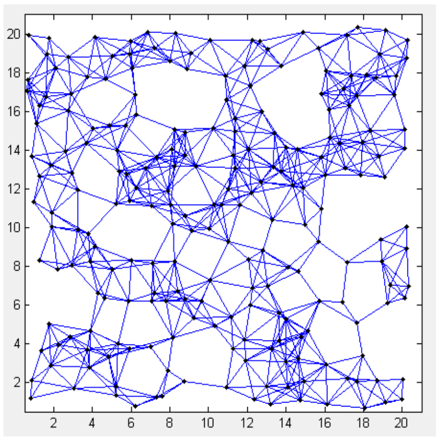




(a) (b)

Figure 3.4 (a) Manually generated topology on a 40×40 grid with obstacles, and (b) Randomly deployed network on 20×20 grid with 40% placement tolerance





(a) (b)

Figure 3.5 (a) Manually generated topology on a 40×40 grid showing connectivity, and (b) Randomly deployed network on 20×20 grid with 40% placement tolerance, showing connectivity with node range 2.3

Figures 3.2 to 3.5 show some network topologies generated using VCSIM. Figure 3.2 (a) shows hexagonal grid topology with 200 nodes deployed on 20 × 20 unit square area. This kind of topology is used in agriculture monitoring or in forests for wild fire detection, where sensors are placed on random trees. In many military applications WSNs are envisioned to be deployed from airplanes. Figure 3.2 (b) shows a random deployment of sensors as in military applications.

Figure 3.3 (a) shows sensor nodes deployed along the street lights. Voids represent the obstacles such as buildings, parks. Figure 3.3 (b) shows the representation of fire alarm sensors in an office or building. Voids in this figure indicate obstacles like a lift or a wall.

Figure 3.4 (a) shows a sensor deployment on a golf course. Bunkers and lakes in golf course are represented by voids. This deployment can be used to study golf ball motion or to improve a play. Figure 3.4 (b) shows random deployment of nodes. Figure 3.5 (a) and (b) shows network deployment with connectivity information. Figure 3.5 (a) shows a deployment in farming for irrigation system. Figure 3.5 (b) shows another random topology. It can be seen that with communication ranges specified by user both the networks are fully connected and no node is isolated. This connectivity information can further be processed to get planar topologies.

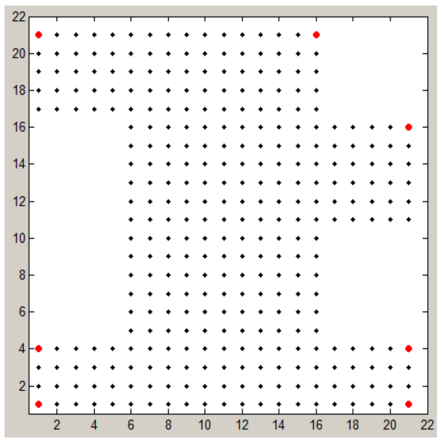
3.1.2 Anchors Placement Methods

Once the required network layout is created user can go ahead for anchor placement. Anchors are essential in generating virtual coordinates for nodes. Anchors are specific nodes which either has additional functionality like GPS receivers, connectivity to base station or are manually configured and placed at specific location. The number of anchors and their placement i.e. the “anchor selection,” is a key factor affecting the performance of VCS based algorithm, including those of routing and topology map generation. Under-placement of anchors results in identical coordinates for different nodes. In general, the routability increases and the likelihood of identical coordinates decrease with the increase in number of anchors. But even over placement of anchors may degrade the performance because of the higher VC generation cost and higher length of address field in packets, causing higher energy consumption due to more transmission length. Simulator also provides a text box which shows node IDs of anchors. In VCSIM anchor placement can be done in three ways:

(a) **Random Anchor Placement:** For this option user specifies a selection criterion called "% of anchors" which indicates percent number of anchors within total number of nodes. Random nodes from the set of available nodes are then selected as anchors. Therefore, if in a 200 node network if user selects 3% anchors, randomly 6 nodes are chosen as anchors.

(b) **Manual Anchor Placement:** User can select particular nodes as anchors either by mouse or by giving approximate coordinates of the nodes. The nearest node to the specified point will be selected as anchor. Anchor selection via coordinates is more convenient when the network is dense.

(c) **ENS (Extreme node search):** This is an anchor selection algorithm proposed in [DhJa11a]. This uses a pair of anchors, which may be randomly or manually selected to generate a directional virtual coordinates. Directional Virtual coordinates are discussed in Chapter 2. Then each node evaluates whether it is a local minima/maxima in DVCs of its h-hop neighborhood. If it is a local minima/ maxima in the neighborhood, the node decides to become an anchor of the network. This scheme usually selects furthest apart nodes as anchors, which is effective in getting unique VCs. In Figure 3.6 (a) and (b) the nodes in 'Red' color show the anchors selected by the ENS algorithm.


(a)

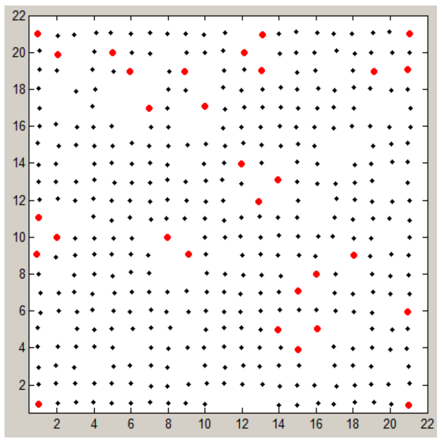

(b)

Figure 3.6 Anchors selected by ENS algorithm for (a) An odd shaped network with 297 nodes, and (b) 20×20 grid based network with 56 nodes mission from random locations

3.1.3 Virtual Coordinates Generation

Once the anchor placement is completed user can press the start button to generate VCs. Two types of VCs generation methods are implemented in simulator. Normal VCs indicate minimum hop distance to each anchor. Directional virtual coordinates are not exactly the hop distances to anchors but are obtained from transformation on VCs as described in Chapter 2. VCs of each node can be seen by using “Node Data Info” button. In [DhJa11a] it is showed that difference in DVC under certain anchor placement could be used for deterministic routing. “Gap Data Info” button shows the difference between DVCs of two neighboring nodes. Figure 3.7 shows the rectangular network after running the simulation. It shows the VCs information and Gap information obtained using “Node Data info” and “Gap Data Info”. Here different colors of lines indicate different gap values. From these lines it can be interpreted that for network shown in Figure 3.7 DVCs of all nodes in one column and/or rows differ from their neighbors with constant value. This information is used in determining node location in Chapter 4.

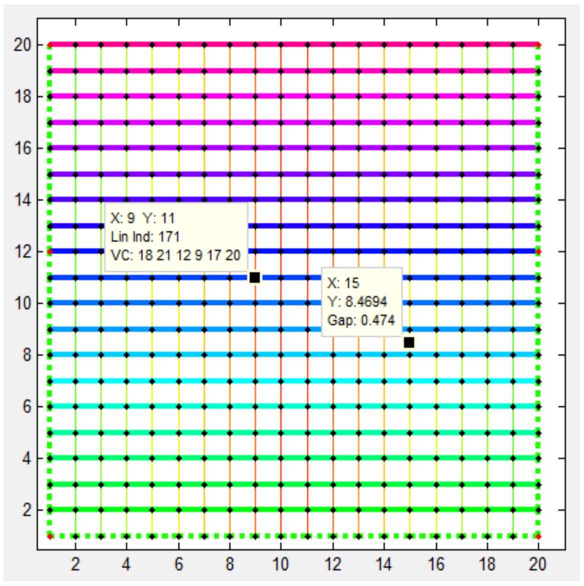
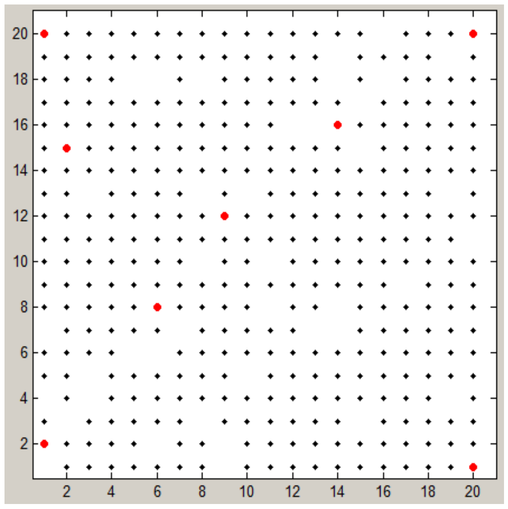


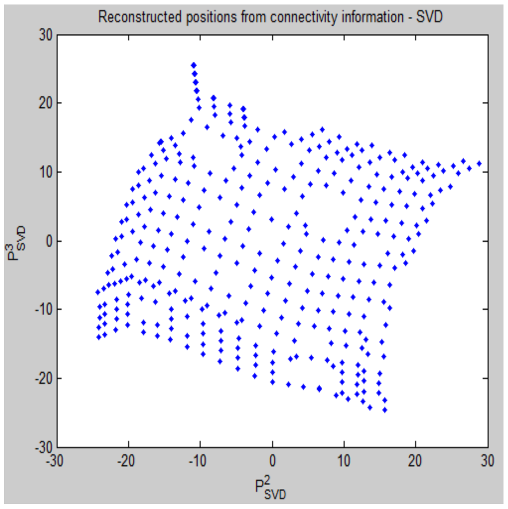
Figure 3.7 Virtual Coordinate information and difference of DVC values as obtained from "Node Data Info" and "Gap Data Info" in VCSIM

3.1.4 Topology Preserving Map Generation

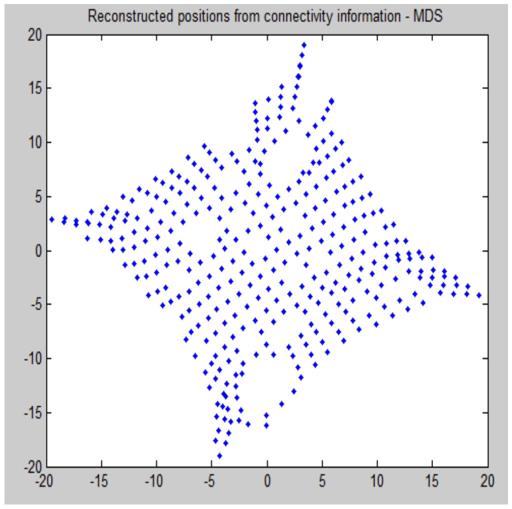
As mentioned in Chapter 1, node location is one of the important techniques in WSN. It is required to report the origin of events, assist group querying of sensors and routing. Due to many reasons such as node cost, size, limited battery power localization remains a hard problem in WSN. Topology preserving maps (TPM) are therefore introduced. Topology preserving maps preserve the neighborhood information without the exact coordinate information. These are useful in routing, network design and network management. Three types of topology preserving techniques are implemented in VCSIM, namely Singular value decomposition (SVD) [DhJa10], multi-dimensional scaling (MDS) [ShRu03] and Directional Virtual Coordinates (DVC) [DhJa11a]. Once the virtual coordinates are generated with good anchor placement user can use the SVD, MDS or DVC option to generate the TPM. MDS is the most computationally expensive algorithm among these three. It uses hop count between all nodes to generate TPM. SVD does not need hop count information within all nodes and uses hop count information only



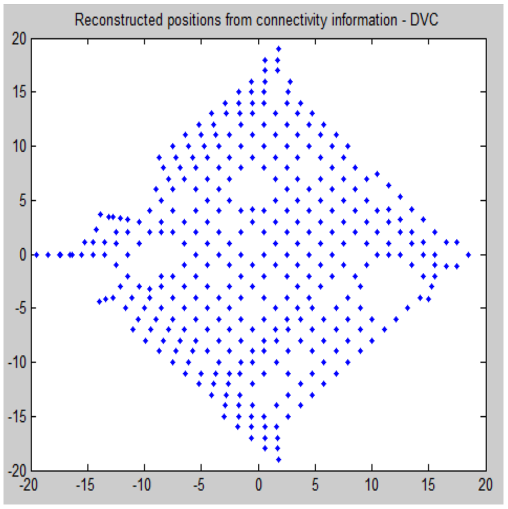
(a)



(b)



(c)



(d)

Figure 3.8 (a) A random network, Topology preserving maps obtained using: (b) Singular Value Decomposition, (b) Multi-dimensional Scaling, and (c) Directional Virtual Coordinates

with anchors. Anchors are usually less in number and therefore SVD is less computationally expensive compared to MDS. The important drawback of SVD is if VCs generated using anchors are identical nodes get overlap. Therefore, anchor placement is important while generating the topology preserving maps. SVD can give the good performance if proper anchor placement is done to obtain unique VCs for each node. DVCs contain directional information. From Eq. 2.1 it can be seen that DVCs increases from one anchor to another. It is zero for in-between nodes i.e. when hop distances to both anchors are equal. If two DVCs making orthogonal angle with each other are obtained, they can be used to generate a 2-D topology. For example, TPMs of the network shown in Figure 3.8 (a) are shown in Figure 3.8 (b), Figure 3.8 (c) and Figure 3.8 (d). Figure 3.8 (b) shows TPM obtained using SVD. Figure 3.8 (c) shows TPM obtained using MDS and Figure 3.8 (d) shows TPM obtained using DVCs.

3.1.5 Topology planarization

Topology planarization is used to remove redundant cross-links in network topology. With this option provided in VCSIM user can run a topology planarization algorithms to obtain a regular rectangular, triangular or a hexagonal topologies from non-planar topologies. Planar topologies are important in WSN as they improve energy conservation, and routing efficiency. Topology planarization is explained in detail in Chapter 5.

3.1.6 Routing

VCSIM currently implements two routing schemes, GPSR and deterministic. VCSIM GUI shows the path packet has followed to reach destination. Simulator also shows number of hops required by packet to reach destination. More about the routing schemes is discussed in Chapter 4.

3.1.7 Undo

VCSIM also provides an "Undo" button. It erases the last changes done and brings back the older state. For example, suppose user created a complex topology and put a void in unintended place. In this case, undo provides an option to revert back to old topology so user can correct the void placement. User can also revert to the old state of anchor placement with this option.

Other than above mentioned features, VCSIM shows the total number of nodes and anchors present in the network. Currently, it also provides file menu option to export the VCs or connectivity information of network.

3.2 Implementation

The implementation of this tool is explained in this section. "Big Picture" of implementation is necessary to understand, modify and add additional capabilities to tool.

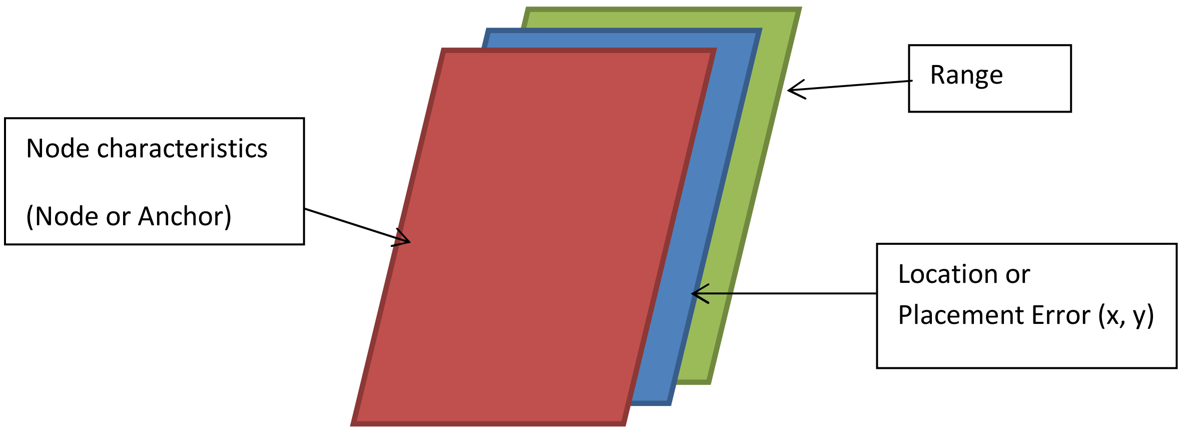


Figure 3.9 In VCSIM, each matrix represent different parameters of sensor nodes

MATLAB is a contraction for "Matrix Laboratory". Therefore as suggested by names, vector operations are more efficiently implemented than looping. WSN is a collection of sensors with similar characteristics. Each sensor node has some parameters like type (anchor or node or different sensing device), range, battery, location of node. Despite all these factors needed to represent a WSN, it is really easy to modify the code and include additional functionalities into the VCSIM. In this simulator, a WSN is represented by a matrix. A presence or type of node is indicated by a matrix element. Therefore, 0 indicates absence of node, 1 indicates presence of node, 2 indicate an anchor. Other properties of sensor are represented by an array of matrix. Figure 3.9 shows an implementation of WSN in VCSIM. In this figure, red color matrix elements represent node type. Blue color matrix elements represent location of nodes and the elements of green color matrix represent the range of nodes. The appendix of this thesis includes the MATLAB code of this tool.

Figure 3.10 shows the simulation flow of VCSIM. User first creates a WSN by defining its parameters; these all parameters are stored in different matrices as explained in previous paragraph. Algorithms are then run as requested by the user. A new parameter of sensor e.g., the available battery power can be added as a separate matrix. Present algorithms may be modified to include new parameters. Similarly, a new algorithm can be implemented as a separate function using available parameters.

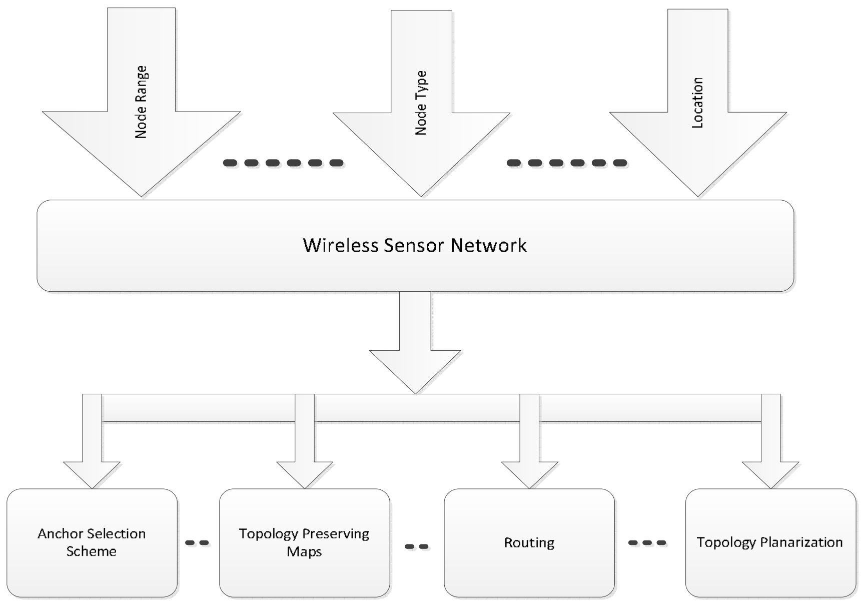


Figure 3.10 Simulation flow of VCSIM

3.3 Usefulness of VCSIM Simulator

The most important advantage of VCSIM is its simple interface to create various WSN topologies. Different methods of anchor selection techniques make this simulator useful for creating VCs. Currently, there are various schemes implemented using Virtual Coordinates like topology preserving maps using singular value decomposition (SVD), or directional virtual coordinates (DVCs). User can make use of these implemented algorithms for comparison. These schemes are not fully tested on different topologies with different anchor selections. Now with this tool it is easy to verify or test these algorithms. With VCSIM user can export virtual coordinates or connectivity matrix of network. This simulator also provides visualization for VCs which can be used in developing tracking algorithms. One can include the additional functions to verify his or her own algorithm.

Chapter 4

Localization and Vector Path Routing in Regular Networks

Localization in WSNs is important for several reasons including identification and correlation of gathered data, node addressing, geographic routing, object tracking and other geographic algorithms. Localization is one of the most fundamental and difficult problems that must be solved for WSN. To overcome the limitation of localization needed for geographic routing VCs are introduced. As discussed in Chapter 2, many types of VCs are proposed using different techniques. For example, minimum hop distance as VCs [CaAb04], or subset of anchors for VCs as proposed in [CaCh05]. One of the techniques is proposed in [DhJa11b] called directional virtual coordinates (DVCs). More about DVCs is explained in Chapter 2. It is showed in [DhJa11a] that these DVCs could be used to get approximate position of node in rectangular networks. [DhJa11a] uses a difference between DVCs of neighbors to know the position of node in rectangular network. In this chapter a method is presented to find the position of node in rectangular, triangular or hexagonal grid using VCs. It is shown that with particular anchor placement, if the length of a shortest path in given topology between a node and anchors is equal to length of a shortest path in full topology (i.e., a topology without any voids) between the same node and anchors, node position in a given topology can be accurately calculated by using VCs. This thesis also presents the work for triangular and hexagonal topologies.

Node Position calculation using VCs:

Rectangular, triangular and hexagonal topologies are shown in Figure 4.1. Here except boundary nodes, each node in rectangular topology is connected to other four nodes, each node in triangular topology is connected to other six nodes and each node in hexagonal topology is connected to other three nodes.

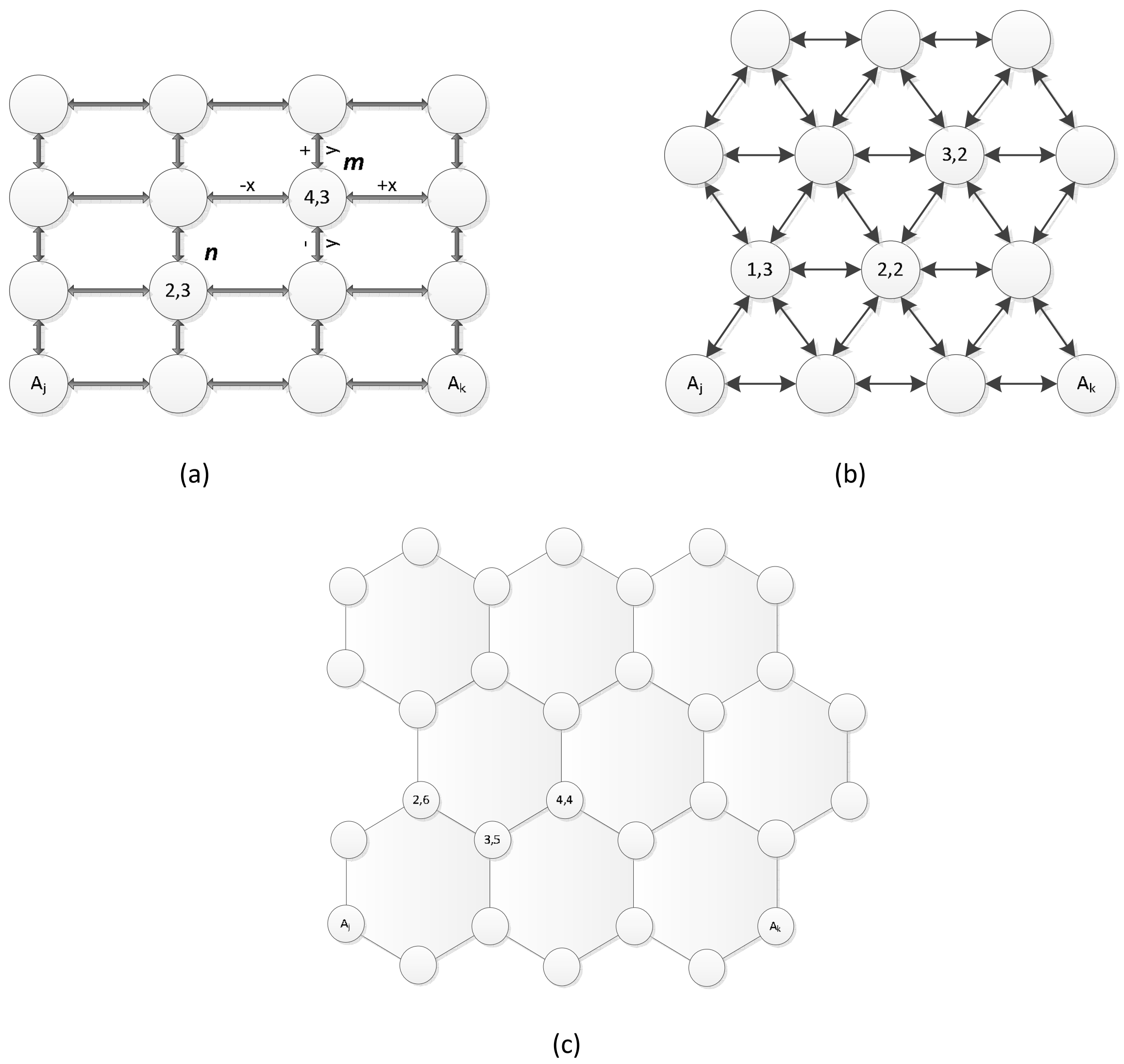


Figure 4.1 Virtual Coordinates of nodes obtained from two anchors $\mathbf{A_j}$ and $\mathbf{A_k}$ in: (a) Rectangular Topology, (b) Triangular Topology, and (c) Hexagonal Topology

Consider sensor nodes are deployed as shown in Figure 4.1. Figure 4.1(a), (b) and (c) shows rectangular, triangular and hexagonal deployment respectively. In Figure 4.1 (a), (b) and (c), virtual coordinates of nodes are obtained from anchors A_j and A_k. From Figure 4.1 (a), it can be seen that only by looking at virtual coordinates of nodes $n(2,3)$ and $m(4,3)$, it is difficult to find their approximate location with respect to anchors or linear relationship between their distances i.e. whether they are 1 hop away on x-axis and 1 hop away on y-axis. Similar conditions occur for triangular and hexagonal topologies (See Figure 4.1 (b) and (c)). It is not straightforward to estimate the location of nodes only from virtual coordinates. Therefore, in this section a coordinate system is presented that can be used to assign addresses to the nodes. The distance between any two nodes can be computed easily if the proposed coordinate assignment schemes are utilized. The addressing schemes for each network are presented in their respective sections. The notations used in this chapter are summarized in Table 4.1.

Table 4.1 Common Notations used in this chapter.

Notations	**Description**
n_i	Node i
n_N	Node's neighbors
A_j	Anchor j
row_{n_i}	Row Number of node i
col_{n_i}	Column Number of node i
$h_{n_iA_j}$	Minimum hop distance between node i and anchor j
$h_{A_iA_j}$	Minimum hop distance between anchor i and anchor j
$D_{n_iA_j}$	DVC of node i.Considering j as first anchor.
$\lfloor n \rfloor$	largest integer not greater than n
$\lceil n \rceil$	smallest integer not less than n

4.1 Rectangular Grid Networks

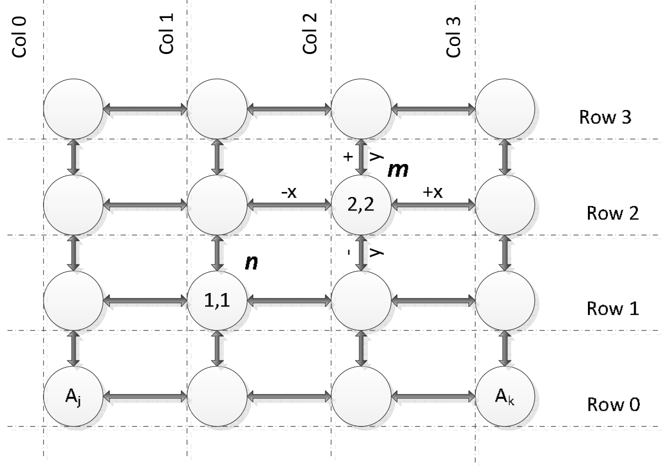
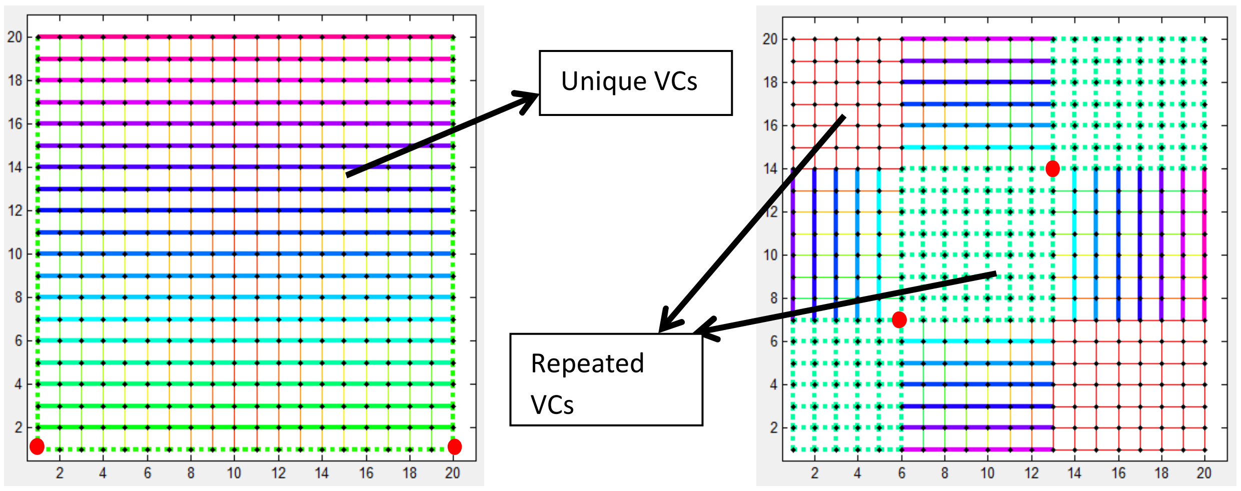


Figure 4.2 Addressing of each node using row and column number in rectangular grid network

Figure 4.2 shows sensor network deployed without void. For address based routing each node should obtain its own physical or virtual coordinates. Virtual coordinates need to be unique in order to uniquely identify each node.

Property 1: The VCs due to anchors A_j, A_k placed parallel to the longest edge and a third non-collinear anchor in a uniform grid network are unique.

Proof: Consider anchors A_j and A_k are placed furthest apart on Row 3 in Figure 4.2. Each node on Row 3 will have unique virtual coordinates. As the distance from one anchor decreases, the distance from another anchor increases. Now as we move the anchor position to row below Row 3, VCs of all nodes on the Row 3 will increase by one to remain unique and the nodes on the same line as anchors will get the VCs one less than the VCs on Row 3. If the same procedure is continued it can be proved that all nodes above Row 0 will have unique VCs due to anchors A_j and A_k. If there are rows below row 0 Nodes symmetrical to Row 0 will have identical VCs due to anchors A_j and A_k. Now if we place any random anchor in one of the half (i.e. either above or below Row 0), non-collinearly to the line joining two initially placed anchors, the symmetrical node in other half will have different hop count due to third anchor. So VCs due to anchors A_j, A_k placed parallel to the longest edge and a third non-collinear anchor will be unique. We verify this by extensive simulations. Figures 4.3 (a) show the region of unique virtual coordinates due to different anchor placement obtained from VCSIM.



(a) (b)

Figure 4.3 Anchor placement: (a) Anchors placed at the bottom corners gives all nodes unique VCs, and (b) In random anchor placement, nodes with identical coordinates increases

4.1.1 Addressing in rectangular grids

A coordinate based addressing is proposed to easily compute the distance between two nodes. In rectangular topology each node can be assigned a row and column number as shown in Figure 4.2. According to coordinate based addressing, nodes n and m shown in Figure 4.2 will have addresses as $(1,1)$ and$(2,2)$. It is easier to get the linear distance between two nodes using these addresses. By knowing coordinate addresses of $n(1,1)$ and $m(2,2)$, it could be determined that these nodes are 1-hop away on x-coordinate and 1-hop away on y-coordinate. As long as there is shortest path between a node and anchors, node can accurately calculate its row and column number from VCs if the anchors placement is assumed to at the extreme bottom corners

$$\boldsymbol{row_{n_i}} = \left(\boldsymbol{h_{n_iA_k}} + \boldsymbol{h_{n_iA_j}} - \boldsymbol{h_{A_jA_k}}\right)/\mathbf{2} \tag{4.1}$$

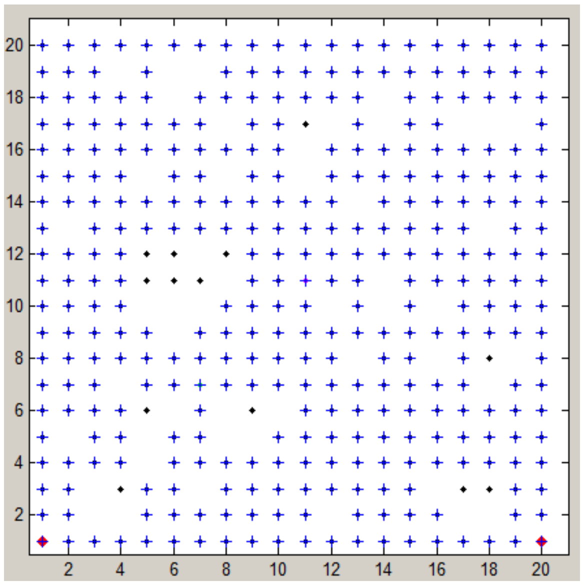
$$\mathbf{col_{n_i}} = \left\lfloor \mathbf{h_{A_jA_k}}/\mathbf{2} \right\rfloor - \left\lfloor \left(\mathbf{h_{n_iA_k}} - \mathbf{h_{n_iA_j}}\right)/\mathbf{2} \right\rfloor \tag{4.2}$$

Consider node n in Figure 4.2. It has VCs$(2,3)$. These VCs can be converted to row and column numbers from Eq. 4.1 and 4.2 as

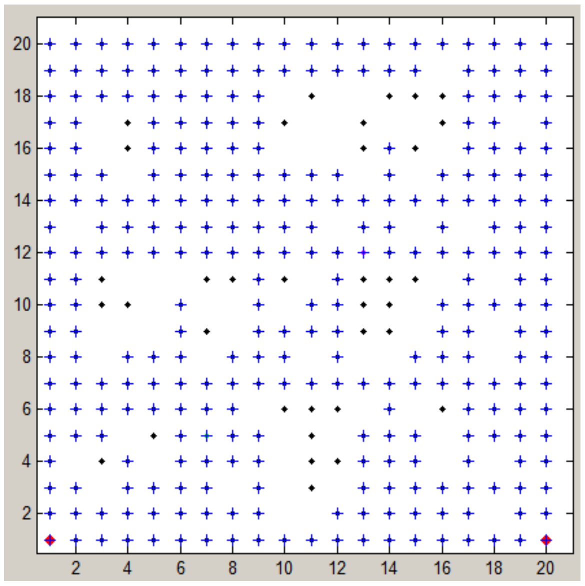
$$row_{n_n} = \frac{2+3-3}{2} = 1$$

$$col_{n_n} = \left\lfloor \frac{3}{2} \right\rfloor - \left\lfloor \frac{(3-2)}{2} \right\rfloor = 1 - 0 = 1$$

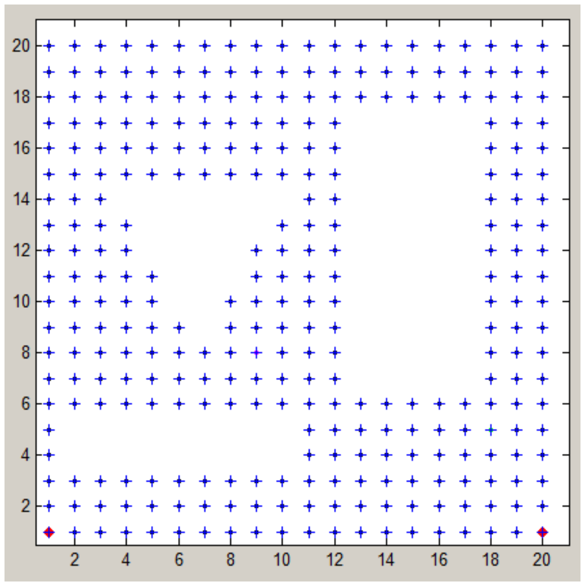
The row and column numbers of nodes will be same if node has identical VCs. Therefore, it is important that each node have unique VCs. Now consider a randomly deployed network with 85% node density as shown in Figure 4.4 (a). Let $\{row_{n_i}\}$ be a row number and $\{col_{n_i}\}$ be a column number. We identify $\{row_{n_i}\}$ and $\{col_{n_i}\}$ based on VCs by applying eq. 4.1 and 4.2. Only 12 out of 349 nodes cannot identify their $\{col_{n_i}, row_{n_i}\}$ position accurately. Figure 4.4 (b) shows a randomly deployed network with 80% node density and Figure 4.4 (c) shows rectangular network with three physical voids. Dot indicates discordant nodes not compatible with their row and column number while '+' sign indicates nodes with accurately identified positions. If anchors placed on long edge are in-between the network, nodes above and below the line joining anchors will have same row and column number. The distinction between lower part and upper part can be resolved by placing the third non-collinear anchor in one of the half as stated in Property 1. Once the positions are calculated, node can use that information for routing. Each node in rectangular network can forward packet to one of the four directions$[+x, +y, -x, -y]$ (see Figure 4.2). Once the locations are known, node is able to determine which neighbor lie in which direction by querying about its neighbors' coordinates.



(a)



(b)



(c)

Figure 4.4 Row and column position identification of node: (a) Random network with 85% node density. 12 out of 349 nodes could not identify their accurate location, (b) Random network with 80% node density. 35 out of 334 nodes could not identify their accurate location, and (c) Rectangular network with three physical voids. All nodes could accurately identify their position

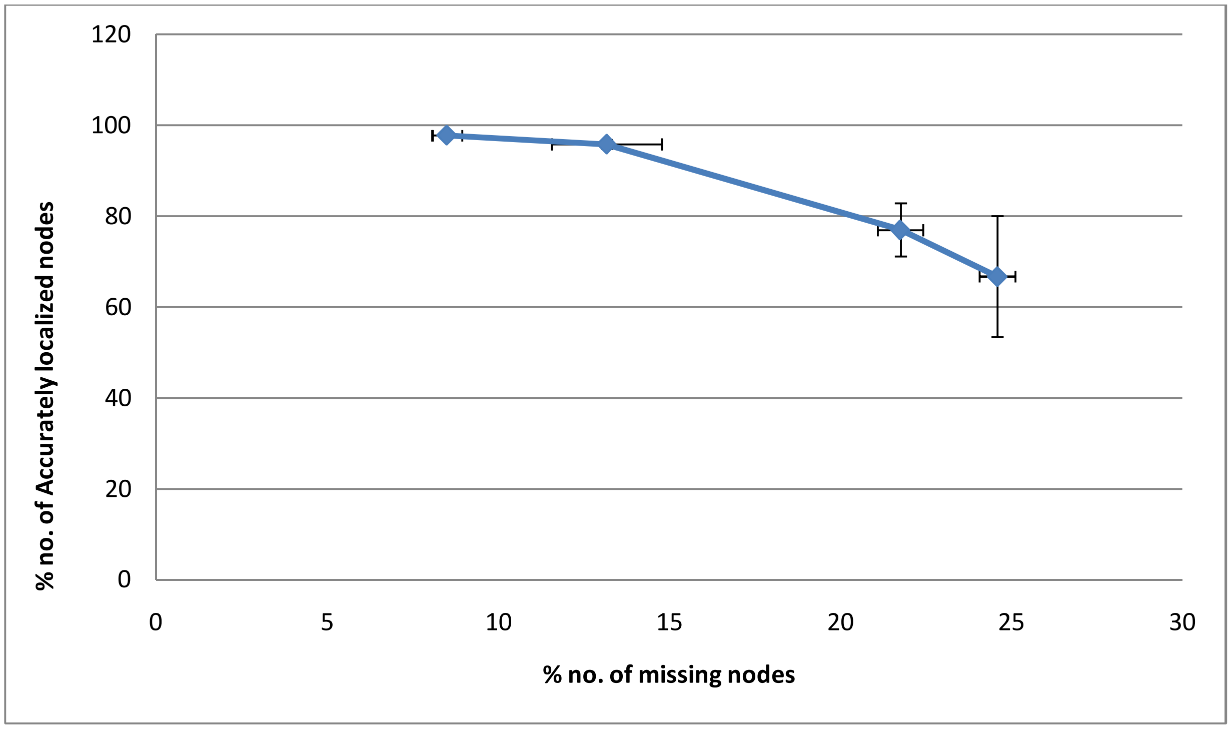


Figure 4.5 % no. of accurately localized nodes vs % no. of randomly missing nodes in rectangular network.

Figure 4.5 shows the plot of % number of accurately localized nodes against % number of randomly missing nodes. If the voids within network are such that shortest distance hop count between the anchor and node doesn't change as compared to full grid, one can accurately localize all nodes. Figure 4.4 (c) shows the network in which obstacle shape does not change the shortest path hop count. In this case all the nodes could accurately identify their location.

4.1.2 Routing in Rectangular Network

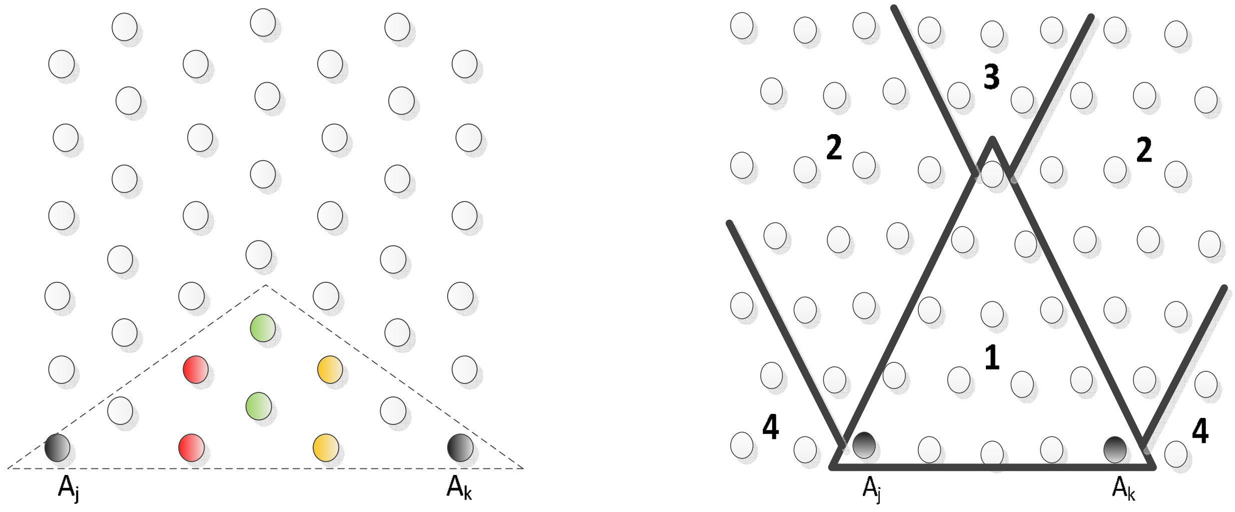
Once the addresses and neighbors direction is known one can use many types of grid based routings like X-Y routing, West-first routing, Negative first routing to name a few. Suppose (row_{n_s}, col_{n_s}) are calculated coordinates of source S and (row_{n_d}, col_{n_d}) are the coordinates of destination D. The magnitude and direction of forwarding can be determined by finding the difference between source and destination coordinates. A simple routing algorithm in full rectangular network (i.e. network without void) can be given as shown in Figure 4.6.

Routing algorithm in full rectangular network:

```
If $(row_{n_s} < row_{n_d})$
        Send packet in positive vertical direction for $(row_{n_d} - row_{n_s})$ times
Else
        Send packet in negative vertical direction for $(row_{n_s} - row_{n_d})$ times
If $(col_{n_s} < col_{n_d})$
        Send packet in positive horizontal direction for $(col_{n_d} - col_{n_s})$ times
Else
        Send packet in negative vertical direction for $(col_{n_s} - col_{n_d})$ times
```

Figure 4.6 Algorithm for routing in full rectangular network

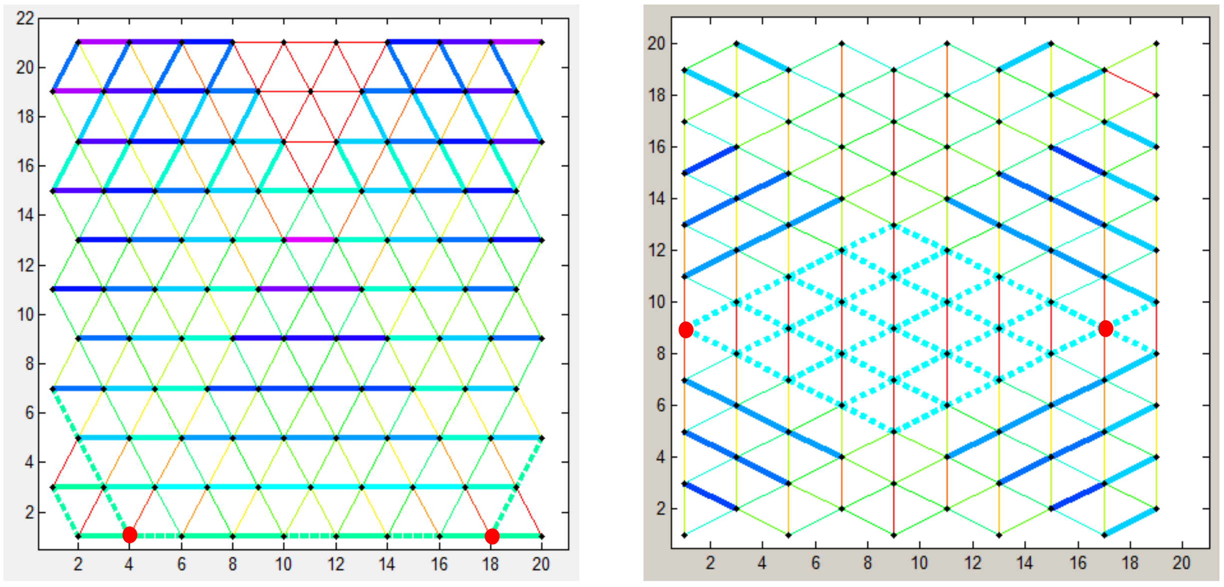
4.2 Triangular Grid Networks



(a) (b)

Figure 4.7 Sensors deployed on triangular grid: (a) Nodes with identical VCs due to anchors placed on zigzag line, and (b) different regions when anchors are placed on straight line

Similar to rectangular grid, in triangular networks anchors placed at the extreme locations on boundary of triangular topology give more nodes with unique VCs. But unlike rectangular grid anchors could be placed in two different ways on an edge in triangular networks (See Figure 4.7 (a) and (b)). Figure 4.7 (a) shows an anchor placement where hop distances between two anchors is 6 hops in a zigzag way. If anchors are placed as in Figure 4.7(a), nodes inside dotted area will have identical Virtual Coordinates. Nodes with same VCs are shown in same color. As the distance between anchors increases for large network, more number of nodes gets identical VCs. Because of identical VCs, unique addresses cannot be obtained. Therefore, anchor placement shown in Figure 4.7 (b) is preferred. Figure 4.8(a) shows the simulation of triangular grid in VCSIM with anchors placed on a straight line and Figure 4.8 (b) shows the identical VCs region when anchors are placed on a zigzag edge as in Figure 4.7 (a).



(a) (b)

Figure 4.8 (a) Unique VCs can be obtained if anchors are placed on straight edge, and (b) Dotted lines showing region of identical VCs

When anchors A_j and A_k are placed as shown in Figure 4.7 (b), the network is partitioned in four regions, namely

Region1. Equilateral Triangle region

Region 2. Deterministic region

Region 3. Zero gap region

Region 4. Unit gap region.

These region numbers are shown in Figure 4.7 (b). Determining the nodes' region is important because VCs to coordinates transformation is different in each region. Nodes have unique VCs in region 1 and region 2. In region 3 nodes are equidistant from both anchors and many nodes can have same VCs so nodes cannot be distinguished uniquely only by VCs. In region 4 the VCs of nodes formed due to both anchors are not independent. Therefore even in region 4 nodes cannot be distinguished uniquely only by VCs. Node i ($h_{n_iA_j}$, $h_{n_iA_k}$) can determine in which region it lies by its VCs by algorithm shown in Figure 4.9

Algorithm for determining nodes' region:

```
$if\ h_{n_iA_j} \le h_{A_jA_k}\ and\ h_{n_iA_k} \le h_{A_jA_k}$
    $then\ n_i \in region\ 1$
$else\ if\ (h_{n_iA_j} > h_{A_jA_k}\ or\ h_{n_iA_k} > h_{A_jA_k})\ and\ 0 < |h_{n_iA_j} - h_{n_iA_k}| < h_{A_jA_k}$
    $then\ n_i \in region\ 2$
$else\ if\ (h_{n_iA_j} > h_{A_jA_k}\ or\ h_{n_iA_k} > h_{A_jA_k})\ and\ (|h_{n_iA_j} - h_{n_iA_k}| == 0)$
    $then\ n_i \in region\ 3$
$else\ if\ (h_{n_iA_j} > h_{A_jA_k}\ or\ h_{n_iA_k} > h_{A_jA_k})\ and\ (|h_{n_iA_j} - h_{n_iA_k}| \ge h_{A_jA_k})$
    $then\ n_i \in region\ 4$
```

Figure 4.9 Algorithm for determining node's region in triangular network

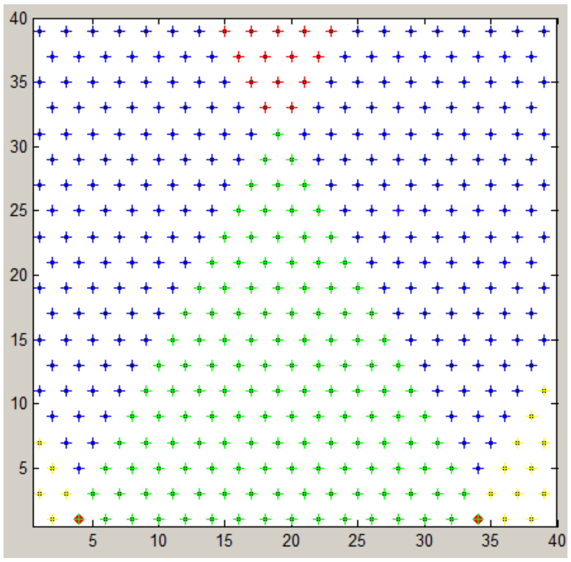


Figure 4.10 Identification of node region in triangular nework using VCSIM, green color indicate region 1, blue color indicate region 2, red color indicate region 3, and yellow color indicate region 4

4.2.1 Addressing in Triangular Network

Coordinates for triangular network are not exactly same as coordinates in rectangular network. To accomplish this task a coordinate system proposed in [NoSt02] is utilized. In this scheme, three axes, x, y and z, parallel to three edge directions and at mutual angle of 120 between any two of them are introduced, as indicated in Figure 4.11. Let i, j, k be the unit vectors in these axes. These three vectors are not independent. They are related by $i + j + k = 0$.

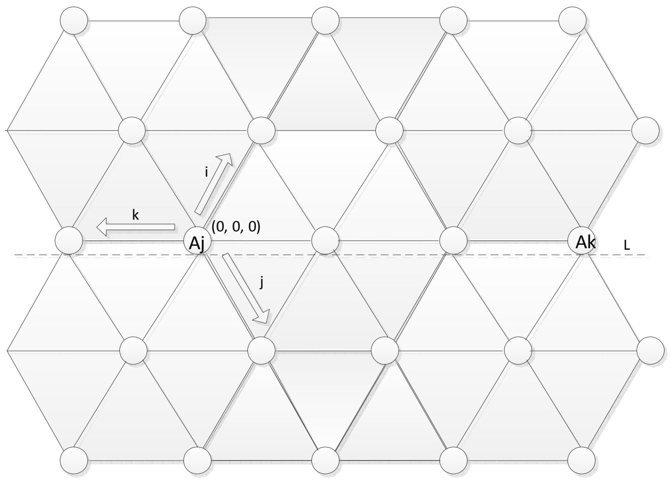


Figure 4.11 Three vectors $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ making angle of 120^{o} with each other are used for addressing

Now, for any node A on the network, if there is a path from origin $(0, 0, 0)$ to node A, and the path has altogether a units of vector i, b units of vector j, and c units of vector k, then the address for node A is $(a, b, c) = ai + bj + ck$. From the definition, such addressing is not unique. For example,$(-1,0,0) = (0,1,1)$ both coordinates represent the same node starting from origin. There are two ways propose to arrive at the unique node addresses in [NoSt02], namely shortest-path and zero-positive form. Considering only shortest path form addressing for this thesis; it is defined as

Definition: A node address $(\mathrm{a}, \mathrm{b}, \mathrm{c})$ is of the shortest path form if

a. At least one component is zero (that is, $abc = 0$) and

b. $ab \leq 0, ac \leq 0, \text{and } bc \leq 0$

Assuming the anchor A_j as origin, the shortest path form address of node n_i $(h_{n_iA_j}, h_{n_iA_k})$ is determined as given in Figure 4.12

Algorithm for determining node's address:

```
If $n_i \in region\ 1$
        Then $(h_{n_iA_j} + h_{n_iA_k} - h_{A_jA_k}, 0, -(h_{A_jA_k} - h_{n_iA_k}))$
If $h_{n_iA_j} < h_{n_iA_k}$ and $n_i \in region\ 2$
        Then $(h_{n_iA_j} - h_{n_iA_k} + h_{A_jA_k}, h_{A_jA_k} - h_{n_iA_k}, 0)$
If $h_{n_iA_j} > h_{n_iA_k}$ and $n_i \in region\ 2$
        Then $(h_{n_iA_k}, 0, -(h_{n_iA_j} - h_{n_iA_k}))$
```

Figure 4.12 Algorithm for determining node's address in triangular network

Therefore, for node n and m shown in Figure 4.2(b) coordinates in shortest path form are

$$n = (2 + 2 - 3, 0, -(3 - 2)) = (1, 0, -1)$$

Which means node n can be reached from origin A after traveling 1 unit vector in $+i$ direction and 1 unit vector in -k direction.

For node $m = (3 + 2 - 3, 0, -(3,2)) = (2, 0, -1)$

Therefore, node m is situated 2 unit vectors in $+i$ direction and 1 unit vector in $-k$ direction from origin.

The path between any two nodes, source $S(x1, y1, z1)$ and destination $D(x2, y2, z2)$ is then calculated as

$$D - S = (x2 - x1, y2 - y1, z2 - z1)$$

If above equation is of the shortest path form, then the length of the shortest path between node S and node D is $|D - S| \; = |x2 - x1| \; + \; |y2 - y1| + |Z2 - Z1|$.

With this addressing scheme each node can uniquely identify other node. But, nodes should know the directions of neighbors for forwarding the packet in correct direction. Section 4.2.1.1 presents the way to identify neighbors' direction.

4.2.1.1 Identifying Neighbor Directions

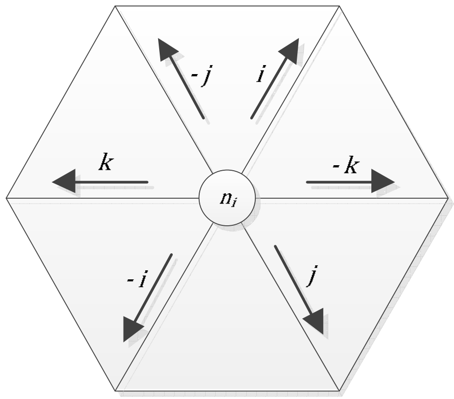


Figure 4.13 Identifying neighbors' directions in Triangular Network

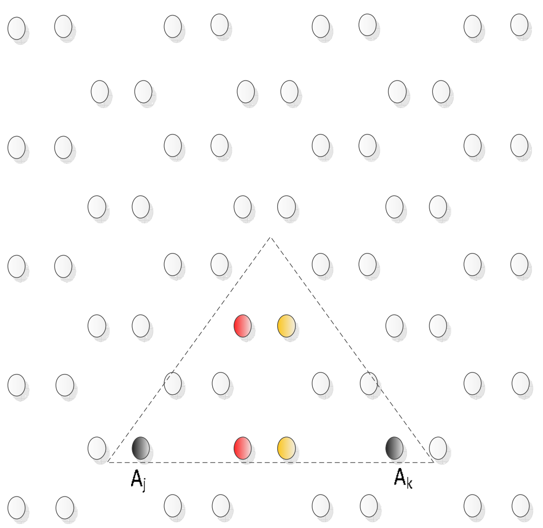
Let i be any node with six neighbors. Each node in triangular network is assumed to have six neighbors. Let the directions of six neighbors are indicated by unit vectors $[+\vec{\imath}, -\vec{\jmath}, +\vec{k}, -\vec{\imath}, +\vec{\jmath}, -\vec{k}]$ as shown in Figure 4.13. Directions of neighbors can either be found using given addressing scheme or by hop count distances. The directions of neighbors N of n_i using hop count distances are obtained as shown in Figure 4.14

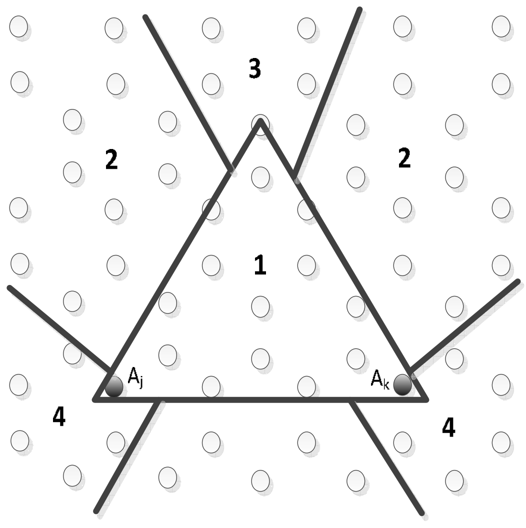
$$\boxed{\begin{array}{l} For\ n_i \in region\ 1 \\ \quad for\ (h_{n_iA_j} + h_{n_iA_k} = h_{n_NA_j} + h_{n_NA_k})\ or\ (min\ (h_{n_NA_j}, h_{n_NA_k}) = \\ h_{n_iA_j}\ or\ min\ (h_{n_NA_j}, h_{n_NA_k}) = h_{n_iA_k}) \end{array}}$$

```
            if $D_{n_{N_1}A_j} > D_{n_{N_2}A_j}$
        Then $-\underset{k}{\rightarrow}$ in $N_1$direction and $+\underset{k}{\rightarrow}$ in $N_2$ direction.
          Else for $h_{n_iA_j} + h_{n_iA_k} > h_{n_NA_j} + h_{n_NA_k}$
            if $D_{n_{N_1}A_j} > D_{n_{N_2}A_j}$
        Then $+\underset{j}{\rightarrow}$ in $N_1$direction and $-\underset{i}{\rightarrow}$ in $N_2$ direction.
          Else for $h_{n_iA_j} + h_{n_iA_k} < h_{n_NA_j} + h_{n_NA_k}$
            if $D_{n_{N_1}A_j} > D_{n_{N_2}A_j}$
        Then $+\underset{i}{\rightarrow}$ in $N_1$direction and $-\underset{j}{\rightarrow}$ in $N_2$ direction.
For $n_i \in region\ 2$, the directions are determined as
          if $h_{n_iA_j} < h_{n_iA_k}$ ? $\theta = A_j : \theta = A_k$
            for $h_{n_i\theta} = h_{n_N\theta}$
              if $D_{n_{N_1}A_j} > D_{n_{N_2}A_j}$
        Then $-\underset{k}{\rightarrow}$ in $N_1$direction and $+\underset{k}{\rightarrow}$ in $N_2$ direction.
            for $h_{n_i\theta} > h_{n_N\theta}$
              if $D_{n_{N_1}A_j} > D_{n_{N_2}A_j}$
        Then $+\underset{j}{\rightarrow}$ in $N_1$direction and $-\underset{i}{\rightarrow}$ in $N_2$ direction.
            for $h_{n_i\theta} < h_{n_N\theta}$
              if $D_{n_{N_1}A_j} > D_{n_{N_2}A_j}$
        Then $+\underset{i}{\rightarrow}$ in $N_1$direction and $-\underset{j}{\rightarrow}$ in $N_2$ direction.
```

Figure 4.14 Algorithm for identifying neighbors' directions in Triangular Network

4.3 Hexagonal Network





(a) (b)

Figure 4.15 Sensor deployed on hexagonal grid: (a) Dotted region shows nodes with identical VCs due to anchors placed on zigzag line, and (b) Different regions when anchors are placed on straight line

Similar to other two types of networks it is found that nodes positioned at extreme location gives unique coordinates. Anchors could be placed in two different ways on an edge in hexagonal network as shown in Figure 4.15 (a) and (b). If anchors are placed as in Figure 4.15 (a) nodes inside dotted area get identical Virtual Coordinates. Nodes with same VCs are shown in same color. As the distance between anchors increases for larger networks more number of nodes gets identical VCs. Because of identical VCs, unique addresses cannot be obtained near anchors. Therefore, anchor placement as shown in Figure 4.15 (b) is preferred. When anchors A_j and A_k are placed as shown in Figure 4.15 (b), similar to triangular network, we divide hexagonal network into four regions, namely

Region1. Equilateral Triangle region

Region 2. Deterministic region

Region 3. Zero gap region

Region 4. Unit gap region.

Algorithm for determining nodes' region

```
if $h_{n_iA_j} \leq h_{A_jA_k}$ and $h_{n_iA_k} \leq h_{A_jA_k}$
   then $n_i \in$ region 1
else if $(h_{n_iA_j} > h_{A_jA_k}$ or $h_{n_iA_k} > h_{A_jA_k})$ and $0 < |h_{n_iA_j} - h_{n_iA_k}| < h_{A_jA_k}$
   then $n_i \in$ region 2
else if $(h_{n_iA_j} > h_{A_jA_k}$ or $h_{n_iA_k} > h_{A_jA_k})$ and $(|h_{n_iA_j} - h_{n_iA_k}| == 0)$
   then $n_i \in$ region 3
else if $(h_{n_iA_j} > h_{A_jA_k}$ or $h_{n_iA_k} \geq h_{A_jA_k})$ and $(|h_{n_iA_j} - h_{n_iA_k}| > h_{A_jA_k})$
   then $n_i \in$ region 4
```

Figure 4.16 Algorithm for determining nodes' region in hexagonal network

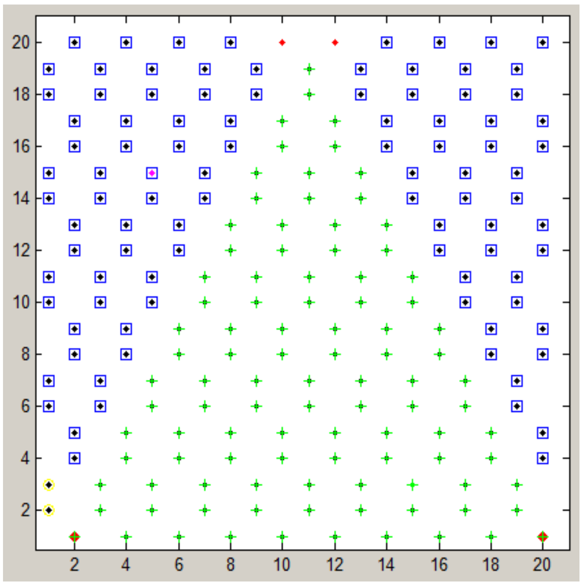


Figure 4.17 Identification of node region in hexagonal network using VCSIM, '+' sign indicate region 1, 'dot' indicate region 3 , and 'square' indicate region 2, 'diamond' indicate region 4

Determining the nodes' region is important because VCs to address transformation is different in each region. Nodes have unique VCs in region 1 and region 2. In region 3 nodes are equidistant from both anchors and many nodes can have same VCs so nodes cannot be distinguished uniquely only by VCs. In region 4 the VCs of nodes formed due to both anchors are not independent. Therefore even in region 4 nodes cannot be distinguished uniquely only by VCs. Any node n_i ($h_{n_iA_j}$, $h_{n_iA_k}$)can determine in which region it lies by its VCs.

4.3.1 Addressing in Hexagonal Network

For addressing in hexagonal networks coordinate system proposed in [ShQa08] is utilized. In this scheme each node is represented by pair (x, y) where x denotes the line number in which the node exists, and y denotes the location of a node on the line. Figure 4.18 shows the addressing in hexagonal network assuming left bottom corner as origin. Node position in terms of row and column number is calculated as shown in Figure 4.19

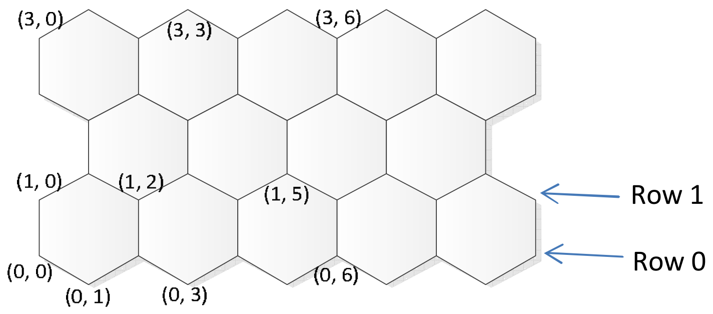


Figure 4.18 Addressing in Hexagonal Network

Algorithm for determining node's address:

```
$if\, n_i \in region\ 1$
        $row_{n_i} = \frac{h_{n_iA_j}+h_{n_iA_k}-h_{A_jA_k}}{2}$
        $col_{n_i} = \left\lfloor \frac{h_{A_jA_k}}{2} \right\rfloor - \left\lfloor \frac{h_{n_iA_k}-h_{n_iA_j}}{2} \right\rfloor$
$else\ if\, n_i \in region\ 2$
    $if\ min\left(h_{n_iA_j}, h_{n_iA_k}\right)$ is odd and two neighbors have $min\left(h_{N_{n_i}A_j}, h_{N_{n_i}A_k}\right) > min\,(h_{n_iA_j}, h_{n_iA_k})$ OR $min\left(h_{n_iA_j}, h_{n_iA_k}\right)$ is even and two neighbors have $min\left(h_{N_{n_i}A_j}, h_{N_{n_i}A_k}\right) < min\,(h_{n_iA_j}, h_{n_iA_k})$
        $row_{n_i} = \left\lceil min\,(h_{iA_j},\ h_{iA_k})/2 \right\rceil$
    $else$
        $row_{n_i} = \left\lfloor min\,(h_{iA_j},\ h_{iA_k})/2 \right\rfloor$
$if\ h_{n_iA_k} > h_{n_iA_j}$ then
        $col_{n_i} = h_{A_jA_k} + row_{n_i} - max\,(h_{n_iA_j}, h_{n_iA_k})$
    $else$
        $col_{n_i} = -row_{n_i} + max\,(h_{n_iA_j}, h_{n_iA_k})$
```

Figure 4.19 Algorithm for determining nodes' address in Hexagonal Network

4.3.2 Routing in Hexagonal Network

Once the node's addresses in terms of given coordinate system and neighbors' directions are known, routing in full hexagonal network can be achieved using following algorithm.

Routing algorithm pseudo code can be given as shown in Figure 4.20

```
While (packet not reached destination)
{
    if ($col_{n_s} != col_{n_d}$)
        moveHorizontal ( )
    else if ($row_{n_s} > row_{n_d}$)
        moveUp ( )
    else if ($row_{n_s} < row_{n_d}$)
        moveDown ( )
}
  moveHorizontal ($col_{n_s}, row_{n_s}, col_{n_d}, row_{n_d}$)
  {
      if ($col_{n_s} > col_{n_d}$)
          Send ($col_{n_{s-1}}, row_{n_s}, col_{n_d}, row_{n_d}$)
      else
          Send ($col_{n_{s+1}}, row_{n_s}, col_{n_d}, row_{n_d}$)
  }
```

```
$\boldsymbol{moveUp}\ (\boldsymbol{col_{n_s}}, \boldsymbol{row_{n_s}}, \boldsymbol{col_{n_d}}, \boldsymbol{row_{n_d}})$
{
        if $(col_{n_s} \in N_i)$
            Send $(col_{n_s}, row_{n_{s+1}}, col_{n_d}, row_{n_d})$
        else
            moveHorizontal $(col_{n_{s\pm1}}, row_{n_s}, col_{n_d}, row_{n_d})$
}
$\boldsymbol{moveDown}\ (\boldsymbol{col_{n_s}}, \boldsymbol{row_{n_s}}, \boldsymbol{col_{n_d}}, \boldsymbol{row_{n_d}})$
{
        if $(col_{n_s} \in N_i)$
            Send $(col_{n_s}, row_{n_{s-1}}, col_{n_d}, row_{n_d})$
        else
            moveHorizontal $(col_{n_{s\pm1}}, row_{n_s}, col_{n_d}, row_{n_d})$
}
```

Figure 4.20 Algorithm for routing in hexagonal grid

4.4 Routing Algorithm in regular networks

Sections 4.1, 4.2, and 4.3 presented a transformation to convert VCs to positional information. Utilizing this coordinate system, one can easily compute distance between two nodes in rectangular, triangular, and hexagonal topologies. Once the addresses and directions of all neighbors are known one can use the routing algorithms for full networks are described in respective sections. But in presence of void routing algorithms should be able to bypass it. Here a routing algorithm is presented which is a modification of full network routing algorithms to bypass the voids. Here a routing path is described in terms of vectors and nodes insert the necessary vector to bypass the obstacle.

VCs can directly be used for greedy forwarding. Some authors have presented this approach. But not always the VCs work very well for greedy forwarding. Face routing is totally based on locations and there is no notion of distance in VCs. Though this thesis does not attempt comparison between any routing algorithms, the point that mainly differentiates greedy forwarding with routing presented in this section is source routing and multipath routing. In a source routing a sender makes all decisions about a routing path of a packet. For source routing to be successful it is assumed that all the nodes have the knowledge about boundary nodes and underlying topology.

Routing Algorithm:

1) Source calculates its position in a network
2) Source attempts to find shortest vector to reach destination by calculating difference between its own and destinations' position.
3) If vector reaches destination without any voids. Source selects that vector as path.
4) Otherwise, source selects a vector with minimum unit vectors in voids.

5) According to various parameters (shortest path, energy of in-between nodes etc.) source selects the final path.

Consider a network shown in Figure 4.21. Source S calculates its position in a network from VCs by using Eq. 4.1 and 4.2. Here the addresses of source and destination are (3, 2) and (7, 10) respectively. Therefore, the Manhattan distance between source and destination is

7- 3 = 4 units in +x-direction

10-2 = 8 units in +y-direction

Henceforth, the arrows [→↑←↓] are used to represent $[+x, +y, -x, -y]$ directions respectively. Therefore in vector form path from (3, 2) to (7, 10) is given by [4→, 8↑].

Total possible paths without void from source to destination in this case would be permutation of [8↑, 4→] vectors. Here for explanation only two vectors are selected namely Vector 1 = [4→, 8↑] and Vector 2 = [8↑, 4→].

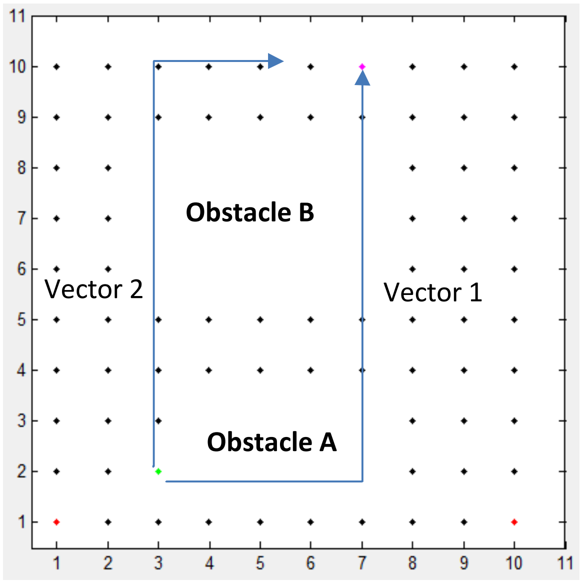


Figure 4.21 Vector 2 is selected as initial vector as it goes through less number of obstacles

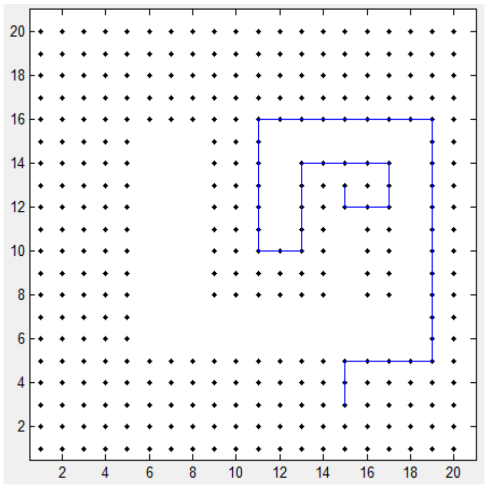
It can be seen that vector 2 goes through only Obstacle B unlike vector 1 which goes through Obstacle A too. As vector 2 goes through less number of voids, it is selected for further path calculation. Complete vector 2 can be initially is given as

Vector 2 = [8↑, 4→] = [↑,↑,↑,↑,↑,↑,↑,↑,→,→,→,→]

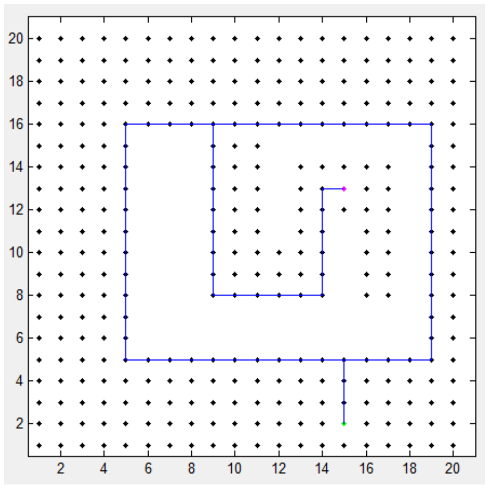
It represents which direction packet should follow to reach destination. It encounters the void at 4th vector and thee void ends at 8th vector. At the start of void, randomly or as per preferences other vector is inserted. The opposite of interested vector is inserted at end of the void. Therefore, if the first vector inserted is ← the vector inserted at end of void position would be →. Suppose the first inserted vector is → then at the end of void position ← vector is inserted. The consecutive opposite vectors are removed. This process is repeated until complete path goes out of void. The final vector for this would be given as

Vector = [↑,↑,↑,→,→,→,→,→,↑,↑,↑,↑,←,↑]

Figures 4.22 to 4.25 show some of routes in rectangular grid network achieved using GPSR and vector routing, assuming each node knows its location:

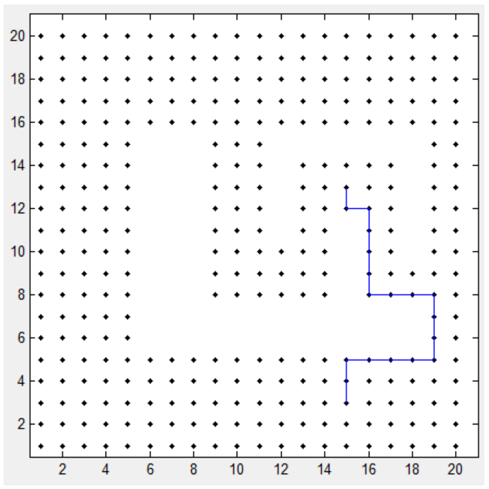


(a)



(b)

Figure 4.22 (a) Vector path routing 47 hops, and (b) GPSR routing 97 hops


(a)

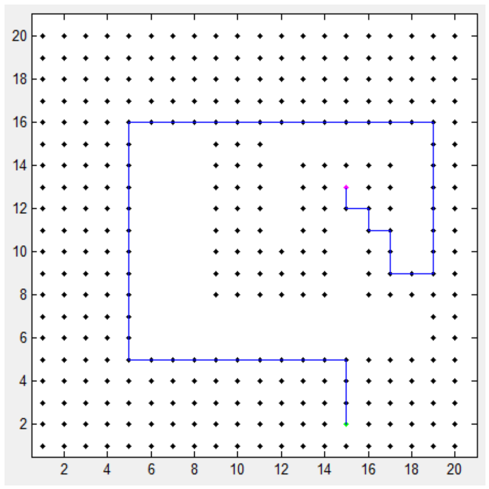
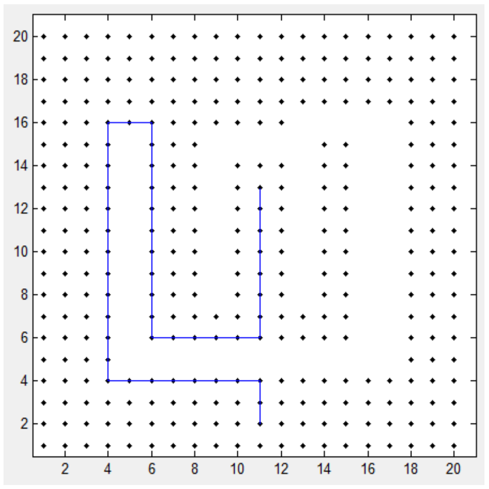

(b)

Figure 4.23 (a) Vector path routing 19 hops, and (b) GPSR routing 53 hops


(a)

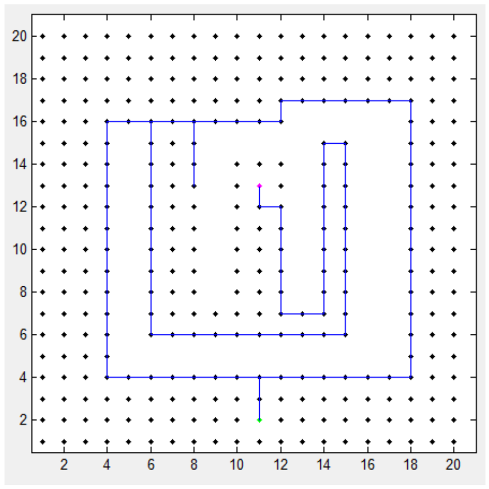
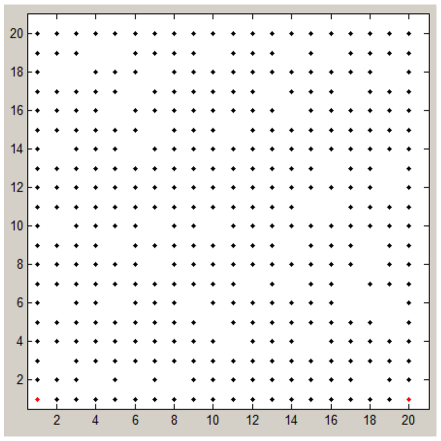
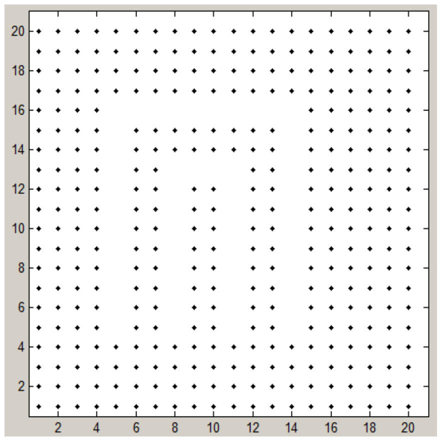

(b)

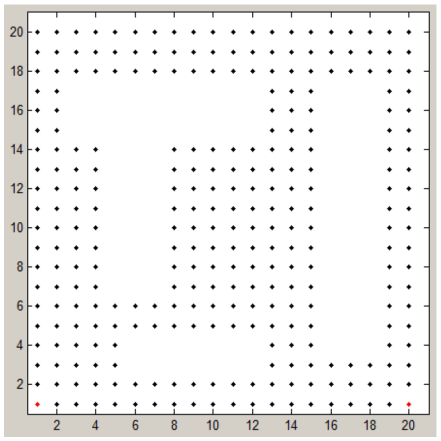
Figure 4.24 (a) Vector path routing 46 hops, and (b) GPSR routing 129 hops



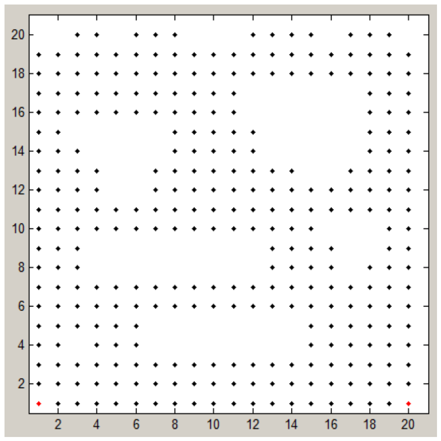
(a)



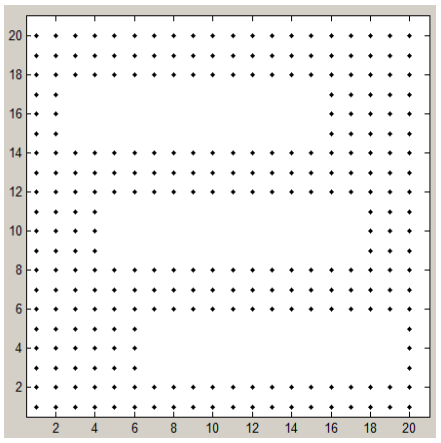
(b)



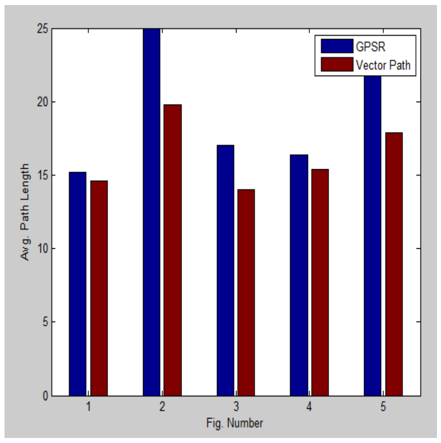
(c)



(d)



(e)



(f)

Figure 4.25 (a) to (e) random network topologies with voids, and (f) Histogram of avg. path length in networks (a) to (e) using vector routing and GPSR

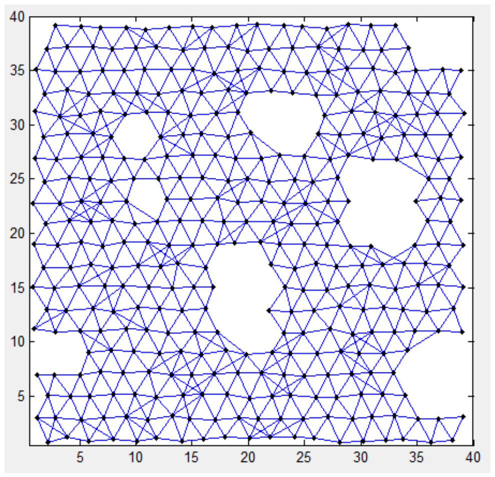
4.5 Conclusion

The chapter presented a transformation from VCs to node position in rectangular, triangular and hexagonal network. It can be seen that for certain anchor placement, if the length of a shortest path in given topology between a node and anchors is equal to length of a shortest path in full topology (i.e., a topology without any voids) between the same node and anchors, node position can be accurately calculated in terms of given coordinate systems. This location results can then be used for routing and location aware services. Multiple paths could be found at source by selecting different preferences (e.g. north first, west first etc.) while inserting the vectors. Results in different random topologies show reduction in average path length for vector routing. This can help in reduced power consumption. Any node in-between source and destination can cache the part of vector and use it later without needing to calculate whole path. Path can be changed by any node by inserting or modifying the vector. Using different paths for same source and destination pair can help in achieving more battery life for nodes. This can also help the network burn energy more equitably and achieve network connectivity for longer time. This routing scheme would be useful in networks with cluster heads where cluster heads are manually placed.

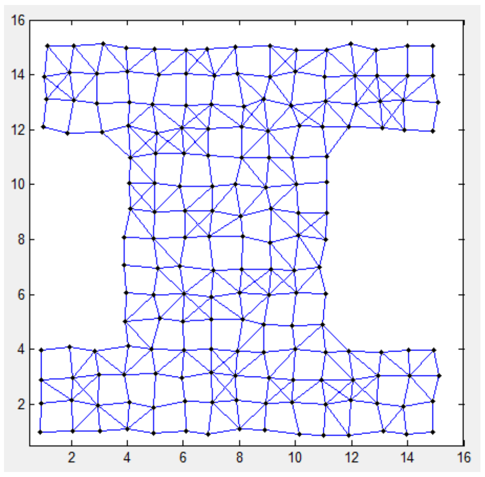
Chapter 5

Topology Planarization

In Wireless Sensor Network (WSN) deployments, wireless sensor nodes should be strategically placed in order to maximize the sensed area while minimizing the number of nodes to be placed in the field. Moreover, wireless radio coverage distances should be taken into account in order to cover large areas. Wireless sensors may be located quite far from each other, allowing lower deployment costs, but it may increase the wireless sensor node power consumption due to the energy needed to reach large distances. Although many factors may affect the position of each wireless sensor node inside the WSN, there are two main ones: the radio coverage area, which allows the sensors to communicate, and, the sensing coverage area, which gives the sensing region. Both types of coverage areas could be affected by the field where the WSN is deployed, but different factors affect each type of coverage. The goal is to maximize the coverage percentage, while coverage holes should be minimized [MuAm10]. Usually the sensing range of a sensor is smaller compared to communication range of a sensor. Therefore, the sensors are placed closed enough to cover the entire area. Most of the research in WSN is presented assuming unit-disk graph model where it is assumed that transmission range of each node is identical in all directions, and nodes within particular range are strongly connected. This is not the case in practice due to non-isotropic radio patterns.



a)



b)

Figure 5.1 Deformed topologies due to constraints in placement of sensors or non-isotropic radios: (a) Sensors placed in agriculture land, and (b) Sensors placed on airport terminal – Closely placed sensors communicate with many other sensor deforming topology.

As shown in Figure 5.1, closely placed sensor nodes along with the non-isotropic radio patterns cause sensor nodes to communicate with other sensor nodes. This increases the interference and causes the non-planar network topology. If we assume each sensor node to be a vertex and connection between them to be an edge, a planar topology means its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other. Network planarization – finding a planar sub graph of the network that contains all the nodes – has been a very important technique in wireless sensor networks for to less interference, conservation of energy, low degree planarity and for many network protocols efficiency. It first became the foundation of various well known routing protocols, including GPSR, GOAFR and several other face-routing protocols. These geographic face routing protocols need sensor network to be planar for better efficiency. Even in flooding, an essential ingredient of many content based routing algorithms, such as DSR or AODV, it is crucial to reduce the number of messages sent. One way to reduce the cost of flooding is to lower the complexity of network by using appropriate topology control mechanisms. It has also been used in numerous other applications, including data-centric storage, network localization, topology discovery, etc. [ZhJi08]. Present techniques of planar topology extraction are often based on accurate node location measurements. The commonly used planarization constructs are: Gabriel Graph (GG), Relative Neighborhood Graph (RNG) and Delaunay Triangulation. These constructs give rules for how to connect vertices placed in a plane with edges based purely on the positions of each vertex's single-hop neighbors. All three techniques provably yield a connected, planar graph so long as the connectivity between nodes obeys the unit graph assumption: for any two vertices A and B, those two vertices must be connected by an edge if they are less or equal to some threshold distance r apart, but must not be connected by an edge if they are greater than r apart. 'r' is the nominal radio range in a wireless network; the notion is that all nodes have perfectly circular radio ranges of radius r, centered at their own positions.

Here the definitions of the GG and RNG are presented for context. The planarization process runs on a full graph, which includes all links in the radio network, and produces a planar sub-graph of the full graph. We assume that each node in the network knows its single-hop neighbors' positions; such neighbor information is trivially obtained if each node periodically transmits broadcast packets containing its own position. Consider an edge in the full graph between two nodes A and B. Both A and B must decide whether to keep the edge between them in the planar graph, or eliminate it in the planar graph. Without loss of generality, consider node A. Both for the GG and RNG, node A searches its single-hop neighbor list for any witness node W that lies within a particular geometric region. If one or more witnesses are found, the edge (A, B) is eliminated in the planar graph. If no witnesses are found, the edge (A, B) is kept in the planar graph. For the GG, the region where a witness must exist to eliminate the edge is the circle whose diameter is line segment AB. For the RNG, this region is the lune defined by the intersection of the two circles centered at A and B, each with radius |AB|. It is shown in two regions in Figure 5.2a. Note that if the network graph violates the unit graph assumption, the RNG and GG can fail in several ways; they can produce a partitioned planar graph [KaKu00], one that contains unidirectional links, and even one that is not planar. Each of these planarization failures can result in a routing failure.

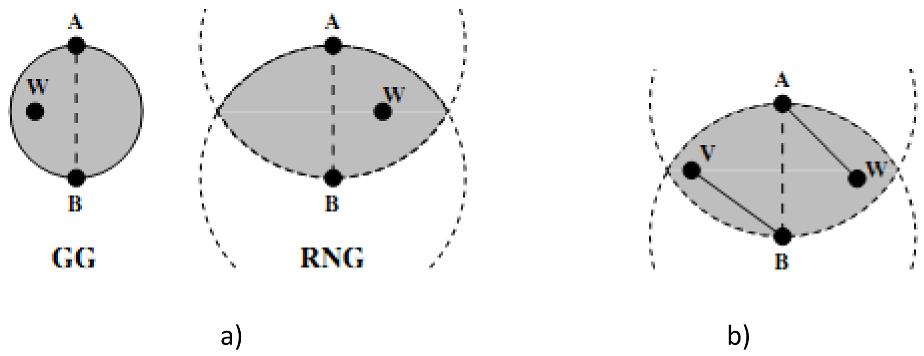
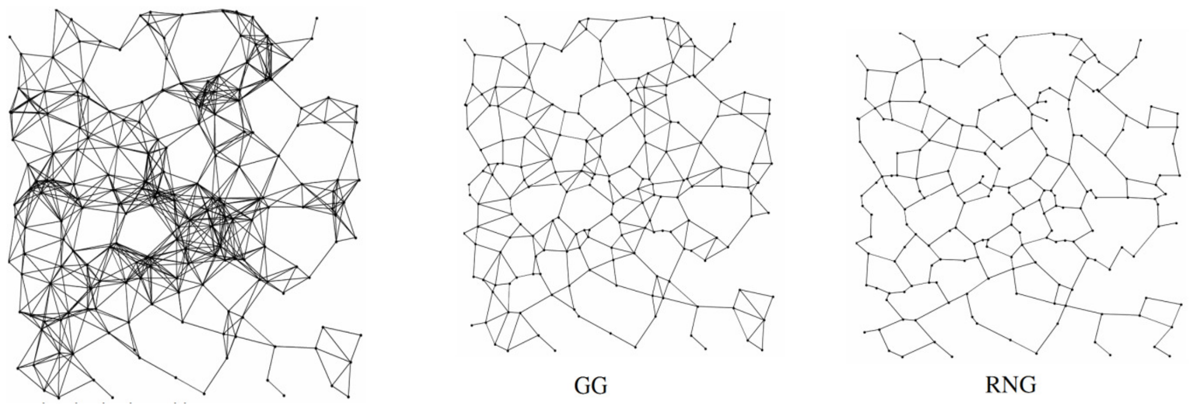


Figure 5.2 (a) GG and RNG method to find connectivity and planarization graph, and (b) Failure in RNG method planarization due to obstacle

An example of a partitioning for the RNG appears in Figure 5.2b. Here, there is no link between A and V, and none between B and W, though these links are shorter than the nominal radio range. Nodes A and B see witnesses W and V, respectively, though neither witness provides transitive connectivity. Both A and B conclude they should remove edge (A, B) in the planar graph, and a partition results. Similar cases are possible in the GG. An example of RNG and GG is shown in Figure 5.3.



(a) (b) (c)

Figure 5.3 (a) A fully connected randomly deployed network in a 20×20 region with 200 nodes and radio range 2.5, and Planarization obtained using: (b) Gabriel Graph method, and (c) Relative neighborhood graph.

A decent amount of research has been done on deployments in wireless sensor networks. A sensor network deployment can usually be categorized as either a dense deployment or a sparse deployment. A dense deployment has a relatively high number of sensor nodes in the given field of interest while a sparse deployment would have fewer nodes. The dense deployment model is used in situations where it is very important for every event to be detected or when it is important to have multiple sensors cover an area. These sensor deployments could be pre-planned or random. A research presented in [ZhHo04, BaYu08] shows that triangular, hexagonal and rectangular are some of the optimal pre-planned topologies. Pre-planned topologies are more efficient in energy consumption, routing etc. Some of the examples of pre-planned topologies are fire alarm sensors, agriculture monitoring, building monitoring, street light controllers etc. But, even in manual placement it is not possible to place sensor on exact location due to many practical constraints. Therefore sensors are placed with some placement tolerance. This causes the non-planar structure within the topology causing crossing edges and interference as shown in Fig 5.1.

This section of thesis presents a topology planarization algorithm for rectangular, hexagonal and triangular network. Unlike other topology control algorithms which assume nodes know their location here the work is presented assuming network is either deployed in one of the three topologies and each node knows or is configured to run software for that topology.

Table 5.1 Common Notations used in this chapter.

Notations	**Description**
n_i	Node i
N_i	Neighbors of node *i*
N_i^2	2- hop neighborhood of node *i*
row_{n_i}	Row Number of node *i*
col_{n_i}	Column Number of node *i*
$h_{n_iA_j}$	Minimum hop distance between node i and anchor *j*
$h_{A_iA_j}$	Minimum hop distance between anchor i and anchor j
$D_{n_iA_j}$	DVC of node i.Considering j as first anchor.
$\lfloor n \rfloor$	largest integer not greater than *n*
$\lceil n \rceil$	smallest integer not less than *n*

5.1 Algorithm to remove cross edges

The algorithm assumes boundary nodes know they are on boundary by some mechanism like boundary detection and each node has 2-hop neighborhood information. In this section we define three terms, namely topology flag bit, incorrectly connected neighbor and correctly connected neighbor. Topology flag bit indicates whether a node is connected to required number of neighbors. For example, in Figure 5.4, nodes are arranged in rectangular topology where except boundary nodes all other nodes should be connected to four other neighbors. But due to placement errors or non-isotropic radio patterns nodes 6, 8, 11, 14, 15, 18, and 19 are connected to more than required number of neighbors. Therefore, these nodes will have their topology bit reset while other nodes will have their topology bit set.

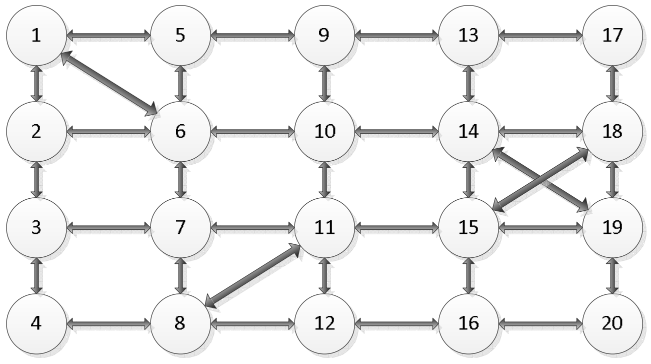


Figure 5.4 Cross-links in non-planar rectangular topology

Similarly, if node is connected to more than six numbers of neighbors in triangular topology and to more than three numbers of neighbors in hexagonal topology its topology bit will be reset. For boundary nodes the conditions will be different in all three topologies i.e. Three number of neighbors in rectangular topology, four number of neighbors in triangular topology and two number of neighbors in hexagonal topology. Incorrectly connected neighbors are defined as neighbors with topology flag bit reset and correctly connected neighbors are defined as neighbors with topology bit set. Therefore, in Figure 5.4, node 6 has nodes 2, 5, 7 and 10 as correctly connected neighbors and node 1 as incorrectly connected neighbor. Node 11 has 7, 10, 12 as correctly connected neighbors and nodes 8 and 15 as incorrectly connected neighbors. Once started algorithm run in three steps.

```
Step 1: Setup the topology flag bit ( )
    (1) If no. of neighbors <= no. of required neighbors
    (2)         Topology flag bit = 1
    (3) Else Topology flag bit = 0
Step 2: Remove the unnecessary links
    (4) If no. of correctly connected neighbors >= no. of required neighbors
    (5)         Remove_incorrectly_connected_neighbor
Step 3: Remove the unnecessary links (Topology dependent)
```

Figure 5.5 Algorithm for removing redundant links

1) Initially each node will either set or reset its topology flag bit according to condition of connected number of neighbors. Therefore, in rectangular topology all except boundary nodes with four neighbors will set their topology flag bit while in other nodes that bit will be reset. Similarly, in triangular topology nodes with six neighbors and in hexagonal topology nodes with three neighbors will have their topology flag bit set. Lines (1) , (2) and (3) in Fig 5.5 shows the step 1.

2) In the next step, if number of correctly connected neighbors i.e. neighbors with topology flag bit set are greater than required number of neighbors, node removes the link with incorrectly connected neighbors i.e. neighbors with topology bit reset.

Therefore, in Figure 5.4 node 6 will remove the connection with node 1 as node 6 have four neighbors with their topology flag bit set.

3) The third step is different for each topology. Only one link removal algorithm doesn't work for all the topologies as the numbers of neighbors are different in all topologies. First, the algorithm for rectangular topology is explained followed by triangular and hexagonal topology.

5.1.1 For rectangular topology

Algorithm for removing cross-links in rectangular topology is shown in Figure 5.6.

```
Step 3: Remove unnecessary links in rectangular network
(1) If $incorrect\_neighbor \in N_i$        $\forall\ i = correct\_neighbor$
(2)         $Remove\_incorrect\_neighbor$
(3) If $N_i^2 \notin N_j$     $\forall\ i = incorrect\_neighbor\ and\ j = correct\_neighbor$
(4)         $Remove\_incorrectly\_connected\_neighbor$
```

Figure 5.6 Algorithm for removing redundant links

a) Lines (1) and (2) in Figure 5.6 shows this step. If incorrectly connected neighbor is neighbor of correctly connected neighbor then remove link to wrongly connected neighbor. Therefore, for node 11 if node 8 or 15 is connected to 7, 10 or 12 then remove that link. Therefore, link 11-8 will be removed.

b) Node will ask for neighbors' neighbor information to the wrongly connected neighbor. If correctly connected neighbor is member of neighbors' neighbor then the node will remove the link to wrongly connected neighbor.

Consider a case of node 14, it has two neighbors with topology flag bit set namely 13, 10 and three nodes 18,19 and 15 with topology flag bit reset. The neighbors' neighbor set of node 18 and 15 include the nodes 13 and 10 which are neighbors of node 14. Therefore, node 14 will remove the link to node 19. Figure 5.7 to Figure 5.11 show planar rectangular topologies obtained in VCSIM using above mentioned algorithm.

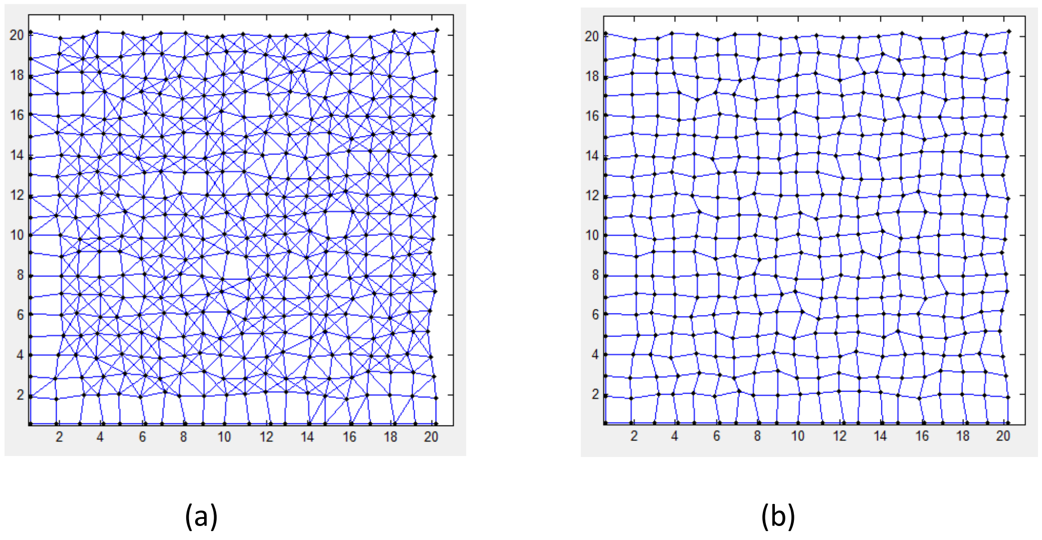
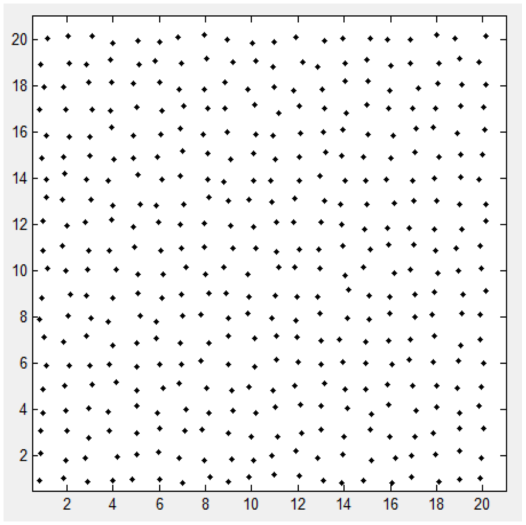
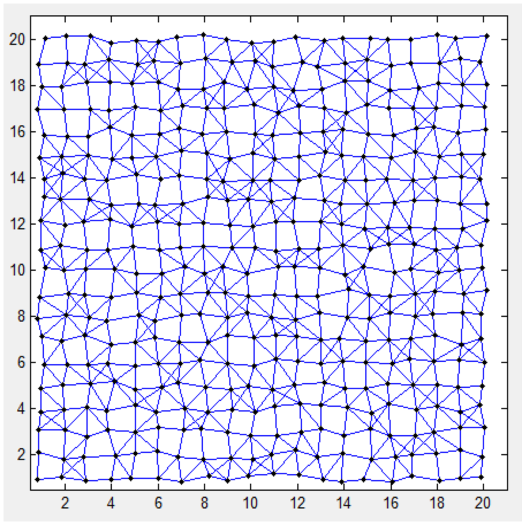


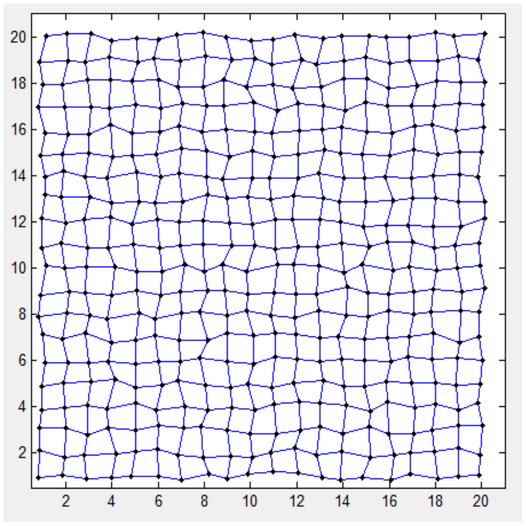
Figure 5.7 Placement Tolerance-20%, and Node range-1.6: (a) Non-planar topology, and (b) Planar topology



(a)

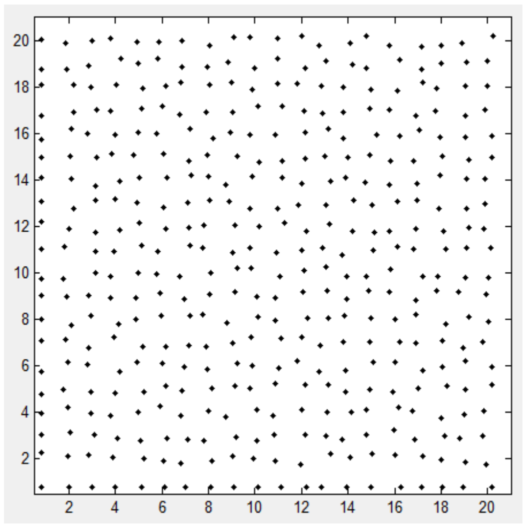


(b)

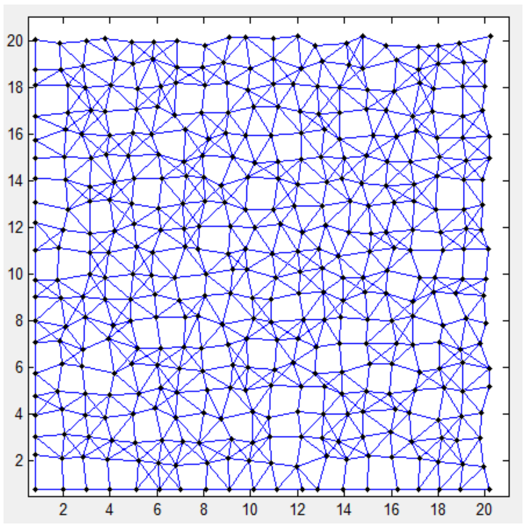


(c)

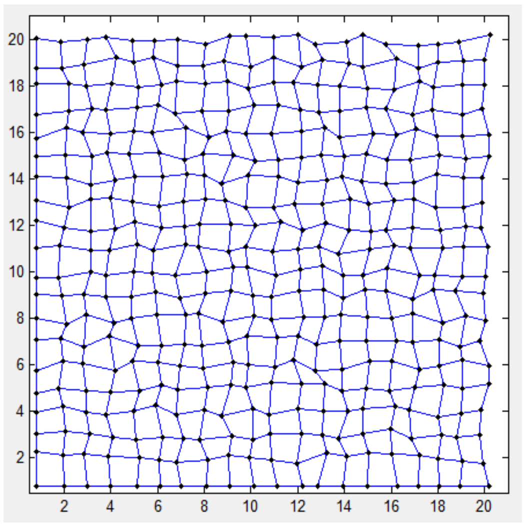
Figure 5.8 Placement tolerance-20%, Range-1.4: (a) Network without connectivity information, (b) Non-planar topology, and (c) Planar topology



(a)



(b)



(c)

Figure 5.9 Placement tolerance-25%, Range-1.5: (a) Network without connectivity information, looks random, (b) Non-planar topology, and (c) Planar topology

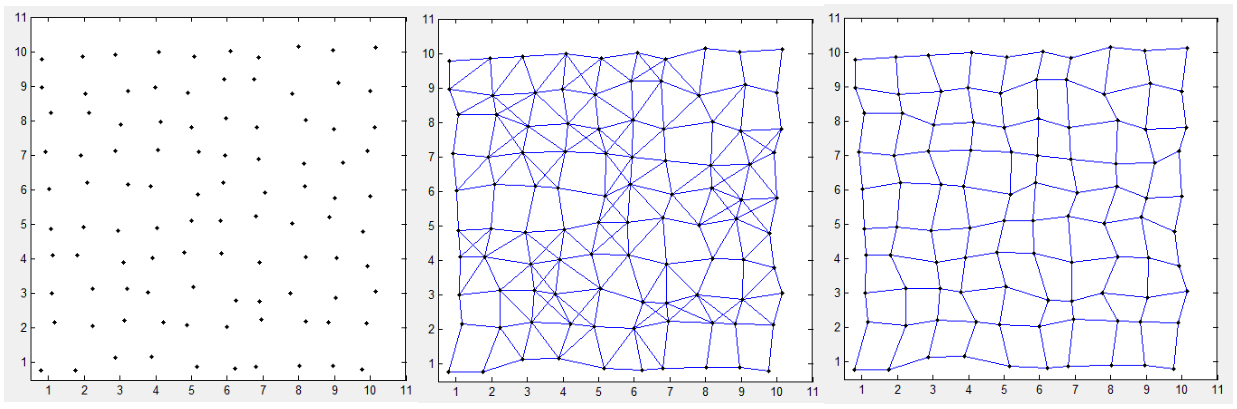

(a) (b) (c)

Figure 5.10 Placement tolerance- 25%, Range- 1.45: (a) Network without connectivity information, looks random, (b) Non-planar topology, and (c) Planar topology

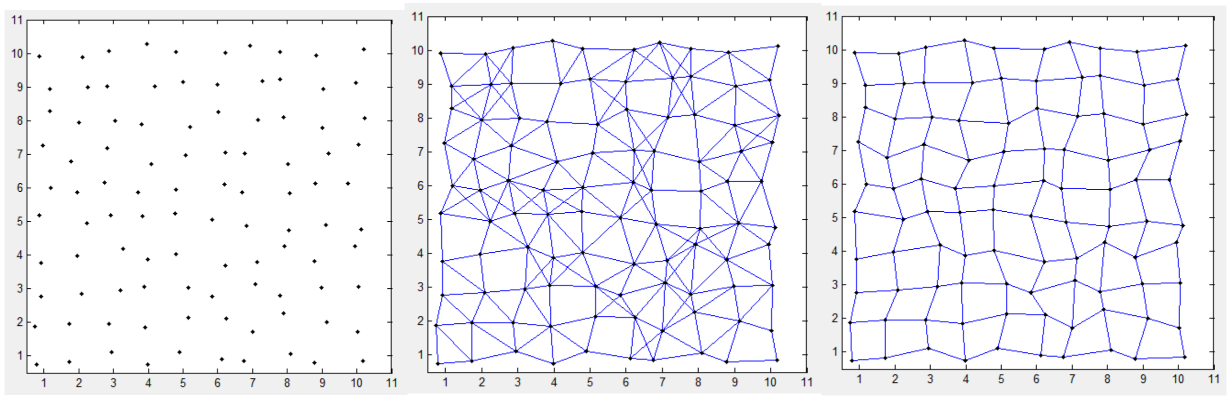

(a) (b) (c)

Figure 5.11 Placement tolerance- 30%, Range- 1.45: (a) Network without connectivity information, looks random, (b) Non-planar topology, and (c) Planar topology

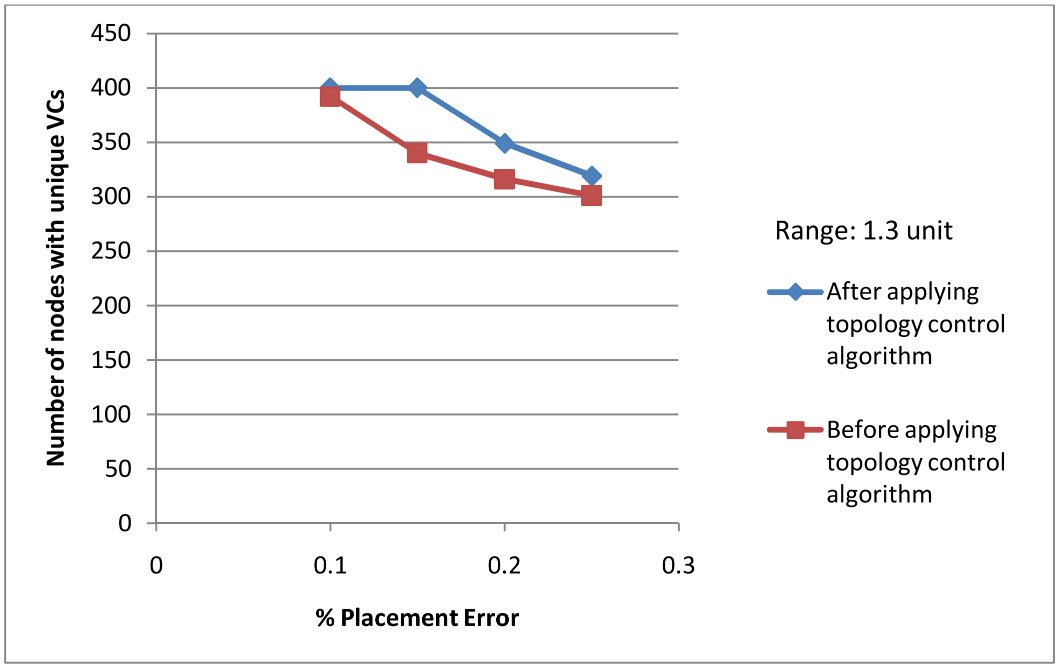


Figure 5.12 Number of nodes with unique VCs are measured before and after applying topology control algorithm to see the effective of algorithm

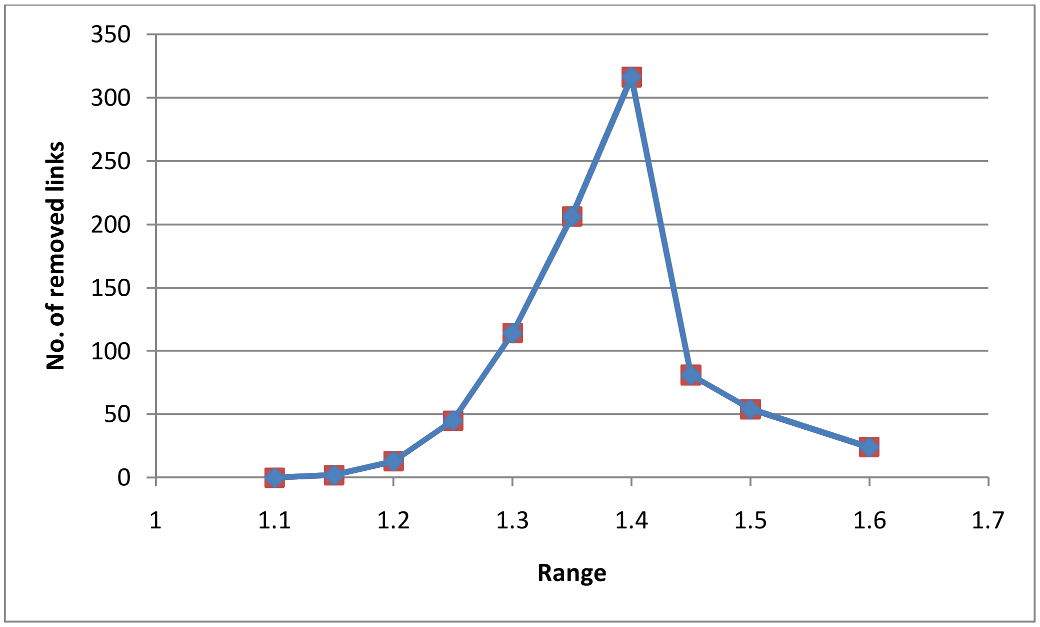


Figure 5.13 No. of removed links are measured against varying communication range to plot the robustness of algorithm

From Figure 5.12, it can be concluded that after applying topology control algorithm more nodes have unique virtual coordinates. Algorithm works linearly till certain communication range. As range increases cross-links increases therefore more number of links has to be removed. After certain communication range too many cross links are formed. Present algorithm cannot remove the cross links in case of congestion. Therefore less number of links are removed after certain communication range. Figure 5.13 shows the robustness of algorithm along the comuunication range.

5.1.2 For Triangular Topology

Third step of triangular topology is different than rectangular topology. Here wrongly connected neighbor of node can be neighbor of correctly connected neighbor. Cross-links in triangular topologies are shown in Figure 5.14. Therefore, the same algorithm as rectangular topology does not work in this network. To remove cross-links in triangular topology algorithm shown in Figure 5.15 is used.

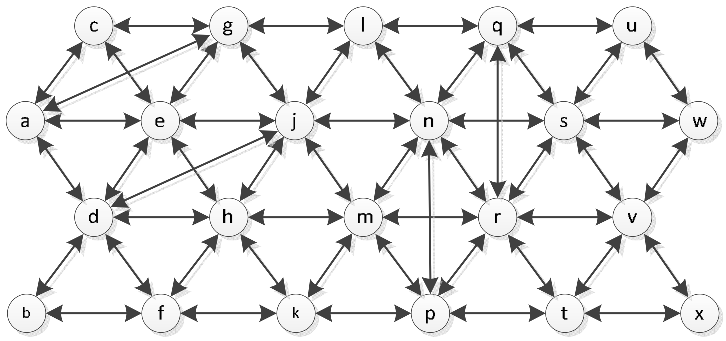


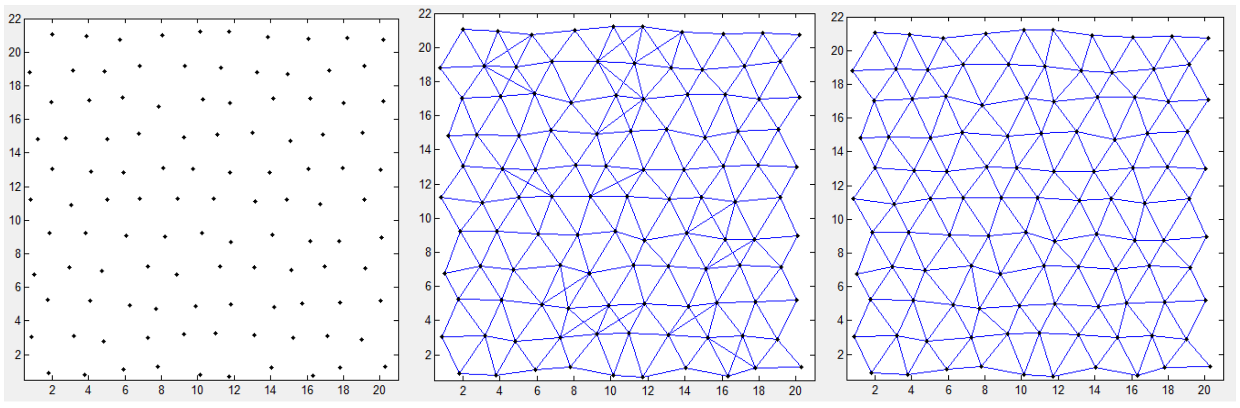
Figure 5.14 Cross-links in non-planar triangular topology

a) If there are only two incorrectly connected neighbors and if the neighbors of incorrectly connected neighbor contain a correctly connected neighbor then notify incorrectly connected neighbors to remove the link between them.

```
Step 3: Remove unnecessary links in triangular network
    (1) If common $N_i^1$(with toplogy flag bit = 1) are neighbors    $\forall\, i = correct\_neighbor$
    (2)          Remove_incorrect_neighbor
    (3) If only one common node (with topology bit set)
    (4)          Remove_incorrectly_connected_neighbor
```

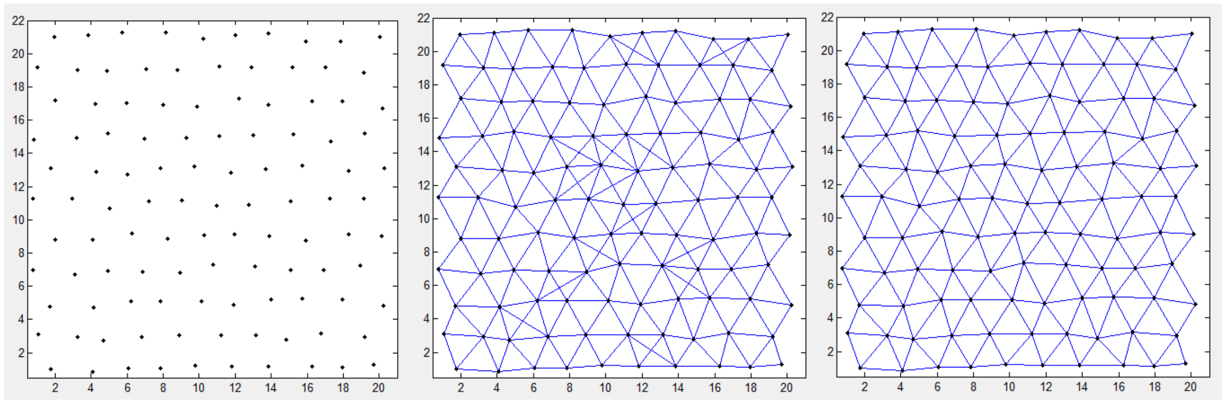
Figure 5.15 Algorithm for removing redundant links in triangular topology

Consider a node h; it has got two neighbors j and d incorrectly connected. Node j has got two incorrectly connected neighbors, node g and d. But it cannot by itself decide which of the two links to remove. When node h sees it has two incorrectly connected neighbors it asks those neighbors if they have any common neighbor which is correctly connected to h, in this case node e. If yes, then node h instructs nodes j and d to remove the link between them.



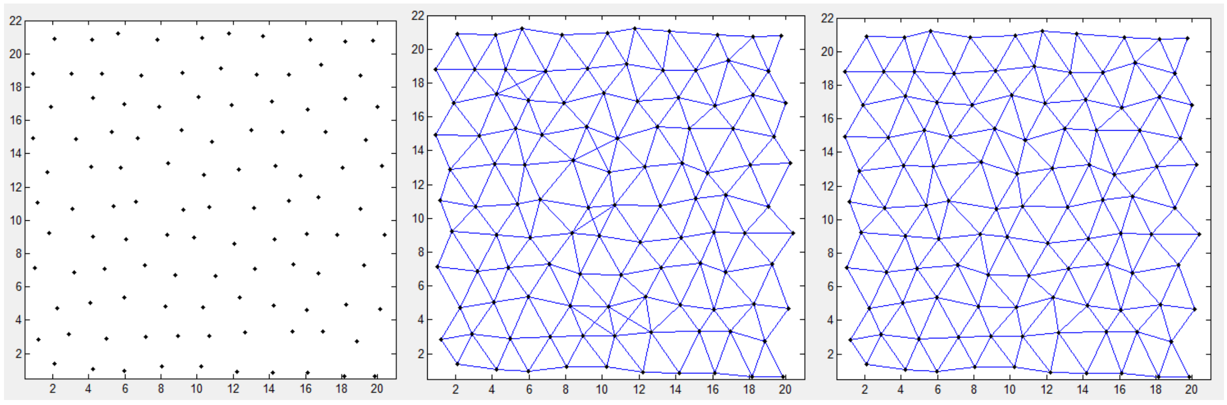
(a) (b) (c)

Figure 5.16 Placement tolerance- 30%, Range- 3.4: (a) Network without connectivity information, (b) Non-planar topology, and (c) Planar topology



(a) (b) (c)

Figure 5.17 Placement tolerance- 30%, Range- 3.3: (a) Network without connectivity information, (b) Non-planar topology, and (c) Planar topology



(a) (b) (c)

Figure 5.18 Placement tolerance- 40%, Range- 3: (a) Network without connectivity information, looks random, (b) Non-planar topology, and (c) Planar topology

b) In case of node s, there are three neighbors which are incorrectly connected, namely node n, q and r. Here node s enquires if these nodes share any common neighbor except those three. In this case node q and r share a neighbor n. Therefore, a link between node q and r is removed. Figure 5.16 to Figure 5.19 show some of the planar rectangular topologies obtained by applying this algorithm.

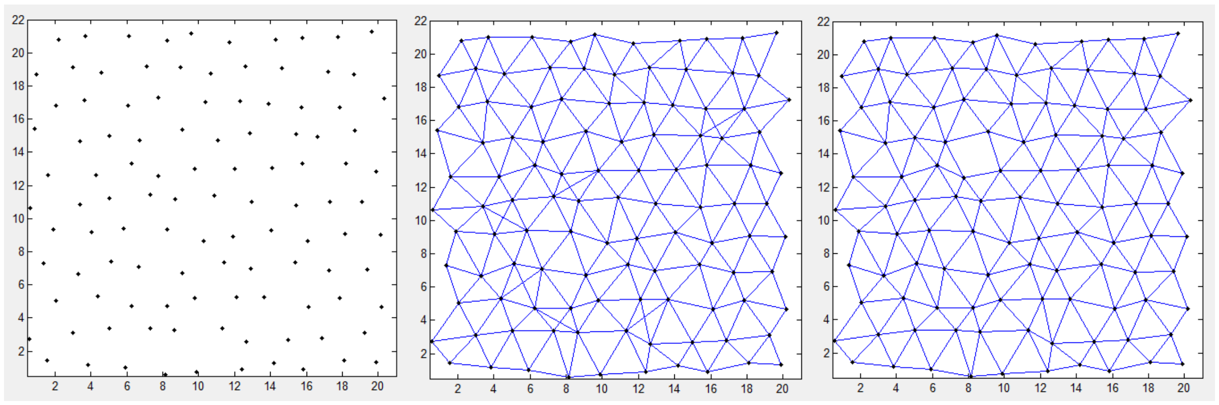

(a) (b) (c)

Figure 5.19 Placement tolerance- 45%, Range- 3: (a) Network without connectivity information, looks random, (b) Non-planar topology, and (c) Planar topology

5.1.3 For Hexagonal Topology

In rectangular topology each node is connected to three other neighbors. Here, no neighbor is connected to other neighbors as in triangular topology or no neighbors share same neighbors as in rectangular topology. Therefore, the algorithms for rectangular and triangular topologies don't work for hexagonal topologies. The algorithm for removing link in hexagonal topology is shown in Figure 5.21.

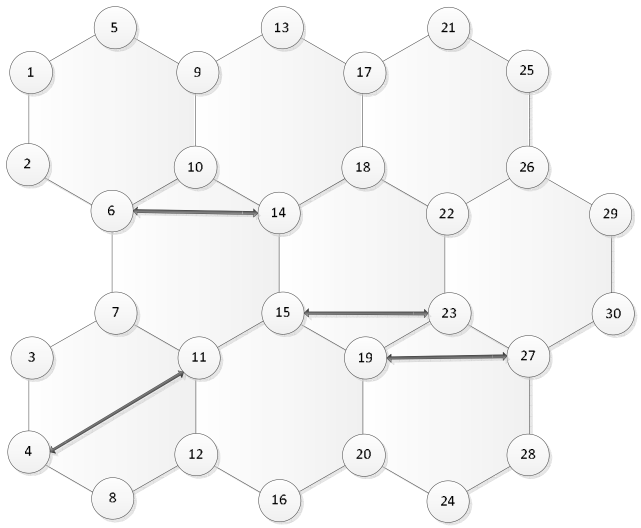


Figure 5.20 Cross-links in non-planar hexagonal topology

```
Step 3: Remove unnecessary links in hexagonal network
(1) If $incorrect\_neighbor \in N_i \quad \forall\, i = correct\_neighbor$
(2)         $Remove\_incorrect\_neighbor$
(3) If $incorrect\_neighbor \in N_i^2 \;\forall\, i = correct\_neighbor$
(4)         $Remove\_incorrectly\_connected\_neighbor$
(5) If $incorrect\_neighbor \in N_i^3 \;\forall\, i = correct\_neighbor$
(6)         $Remove\_incorrectly\_connected\_neighbor$
```

Figure 5.21 Algorithm for removing redundant cross links in hexagonal network

a) The redundant link is removed only if incorrectly connected neighbor is connected to correctly connected neighbor. Therefore, in Figure 5.20 link to node 6 will remove the link to node 14 as it is a neighbor of correctly connected neighbor 10.

b) The link is removed if incorrectly connected neighbor is neighbor of correct neighbor of correctly connected neighbor. In Figure 5.20, node 11 has two wrongly connected neighbors node 4 and node 15. Therefore, it cannot decide which link to remove. The link to node 4 is removed as it is connected to correct neighbor of correctly connected neighbor. In this case, node 12 is correctly connected to node 11 and node 8 is a correct neighbor of node 12. As node 4 is connected to node 8, link to node 4 is removed.

c) The link is removed if incorrectly connected neighbor is neighbor of correct neighbor's neighbor of correctly connected neighbor. In Figure 5.20, Node 19 or node 15 cannot decide initially which link to remove as three or more neighbors indicate they are wrongly connected. But node 19 has got one correctly connected neighbor, node20. It asks node 20 about its correctly connected neighbors correct neighbors which include node 28. As the wrongly connected node 27's neighbor is node 28. Link to node 27 is removed. Figures 5.22 and 5.23 show some of the planar hexagonal topologies obtained in VCSIM.

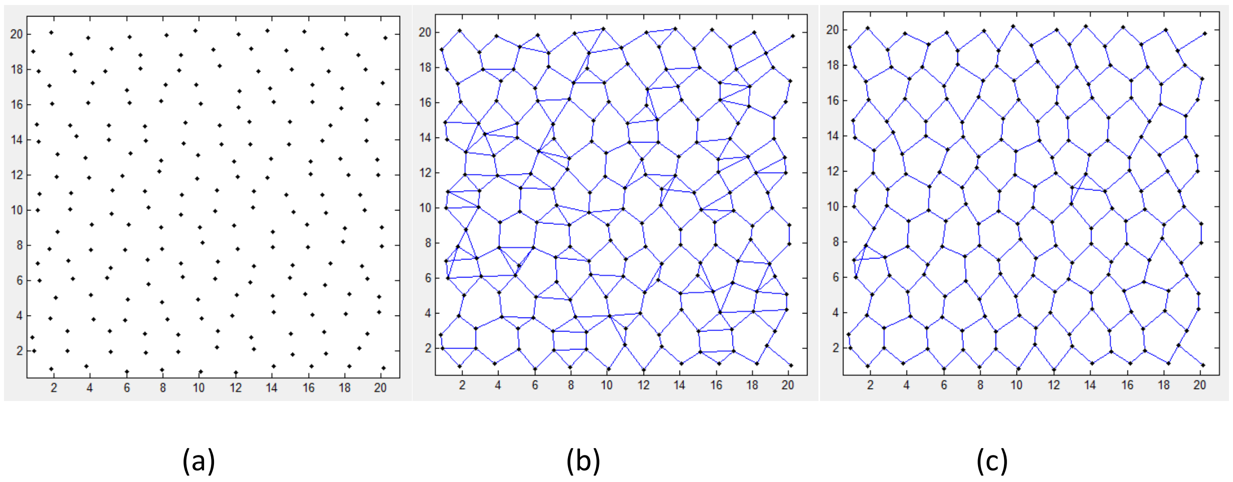


Figure 5.22 Placement tolerance- 25%, Range- 1.9: (a) Network without connectivity information, (b) Non-planar topology, and (c) Planar topology

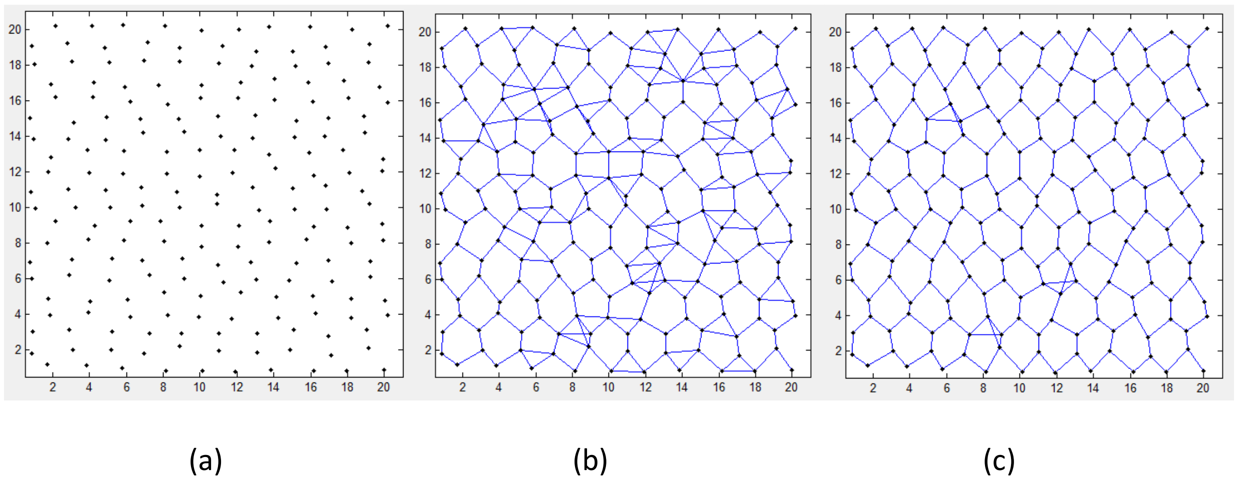


Figure 5.23 Placement tolerance- 30%, Range- 1.9: (a) Network without connectivity information, (b) Non-planar topology, and (c) Planar topology

5.2 Conclusion and Summary

This section presented a network planarization algorithm for manually deployed rectangular, triangular and hexagonal topologies. Unlike other algorithms, this did not use location or distance information between nodes. Also, with some of planarization algorithms like Delaunay triangulation planarization of rectangular topology is not possible as it uses three points within the circle to establish connectivity which means node can be connected to maximum three numbers of neighbors. Good planarization would help in getting the correct location from virtual coordinates, therefore, allowing better routing schemes. Planarization of regular topologies will also help in other routing protocols based on topology hierarchy (e.g. LEACH) where certain nodes will act as cluster heads and topology of cluster heads will have regular topologies. With this algorithm cluster heads did not have to know their locations only the logical links.

Chapter 6

Summary, Conclusion and Future Work

6.1 Summary and Conclusion

Due to large scale and other factors involved simulation is very important aspect for studying wireless sensor networks. Though many types of general purpose and some WSN specific simulators are available with many protocol features, no tool is available with intuitive interface for virtual coordinate space simulation in WSN. VCSIM is a wireless network simulator designed specifically for simulating virtual coordinate space of wireless sensor networks. It can be used to study behavior in large scale networks. VCSIM can facilitate the creation of network with different parameters like range, size, shape, distribution etc. It can be used for placement of anchors, to visualize connectivity within nodes, obtain topology preserving maps, and generate minimum hop distance and directional VCs. With its GUI and different available options for VCs, VCSIM will help in studies such anchor selection algorithms, routing and tracking in WSNs using VCs.

Chapter 4 presented the transformation of VCs to position information. It is showed that with certain specific anchor placements, a location of a node can be accurately approximated, if the length of a shortest path in given topology between a node and anchors is equal to length of a shortest path in full topology between the same node and anchors.. These locations are obtained without any special hardware. This positional information can then be used for location aware services or routing. It is found that average path length achieved using deterministic routing is less than greedy forwarding. This will increase the network efficiency in regular networks by reducing power consumption and/or using different paths. This method of localization and routing using VCs in regular networks can be particularly useful in three dimensional surface topologies where greedy forwarding is more prone to failure due to 2D local minimum.

Chapter 5 showed the planarization of rectangular, hexagonal and triangular topologies. Even manually deployed networks have non-planar topologies due to placement tolerance and non-isotropic radio patterns. Most of the present network planarization algorithms are based on location information of node. In common deployment of manual networks, sensors may not have location information. Even if the location information is available with present localization schemes such as GG, RNG or Delaunay, it is not possible to get rectangular grid like planarization. Network planarization without location information is presented in Chapter 5. The results presented in this thesis show that with up to 30%, 45%, and 30% placement errors in rectangular, triangular and hexagonal topologies respectively we can get the good planar topologies without location information. In planar topologies more nodes have unique VCs compared to VCs generated in non-planar topology. Good planarization would help in getting the accurate position from virtual coordinates as described in Chapter 4, therefore, allowing better routing schemes.

6.2 Future Work

VCSIM can be expanded to include other VCS based algorithms like Convex Subspace Routing (CSR). Currently, simulator does not have more realistic radio transmission model. Future versions may include more realistic radio transmission models with path loss or anisotropic radios etc. Other ongoing research like tracking in VCs could also be implemented. Present version of simulator does not allow creation or visualization of 3-D network topologies. Future versions may include 3-D network creation along with routing in 3-D topologies.

The localization method presented in Chapter 4 does not work if the length of a shortest path in given topology between a node and anchors is not equal to length of a shortest path in full topology between node and anchor. In that case the algorithm could be extended to find 2nd order approximation from correct 1st approximated nodes where correct 1st order approximated nodes will assign themselves as anchors and a location of discordant nodes can be calculated. The result can also be extended to find the different zones in a network from same pair of anchors and then the packet could be forwarded to different zones. For identical VCs in hexagonal and triangular network VCs should be studied for more than 2 anchors. Network planarization algorithms can also be studied for 3-D topologies.

References

[AkYo05] K. Akkaya and M. Younis, “A Survey of Routing Protocols in Wireless Sensor Networks Ad Hoc Network,” Elsevier, 2005, vol. 3, no.3, pp. 325–349

[AlKa04] J.N. Al-Karaki and A.E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey," IEEE Wireless Communications, 2004, vol.11, no.6, pp. 6-28

[BaTa05] J. Bachrach and C. Taylor, “Localization in Sensor Networks,” in “Handbook of Sensor Networks: Algorithms and Architectures”, 2005, 1st ed.

[BaYu08] X. Bai, Z. Yun, D. Xuan, T. H. Lai, and W. Jia, “Deploying Four-Connectivity and Full-Coverage Wireless Sensor Networks,” IEEE INFOCOM, 2008

[BrEs02] D. Braginsky and D. Estrin, “Rumor Routing Algorthim for Sensor Networks” Proc. of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, 2002, pp. 22-31

[CaAb04] Q. Cao and T. Abdelzaher, "A Scalable Logical Coordinates Framework for Routing in Wireless Sensor Networks," Proc. of the 25th IEEE International Conference on Real-Time Systems Symposium, 2004, pp. 349- 358

[CaCh05] A. Caruso, S. Chessa, S. De, and A. Urpi, "GPS Free Coordinate Assignment and Routing in Wireless Sensor Networks," INFOCOM 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 2005, vol.1, pp. 150- 160

[DhJa09] D. C. Dhanapala and A. P. Jayasumana, "CSR: Convex Subspace Routing Protocol for Wireless Sensor Networks," Proc. of the 34th IEEE Conference on Local Computer Networks, 2009, pp. 101-108

[DhJa10] D. Dhanapala and A. P. Jayasumana, "Topology Preserving Maps from Virtual Coordinates for Wireless Sensor Networks," Proc. of the 35th IEEE Conference on Local Computer Network, 2010, pp. 136-143

[DhJa11a] D. C. Dhanapala and A. P. Jayasumana, "Anchor Selection and Topology Preserving Maps in WSNs — A Directional Virtual Coordinate Based Approach," Proc. of the 36th IEEE Conference on Local Computer Networks, 2011, pp. 571-579

[DhJa11b] D. C. Dhanapala and A. P. Jayasumana, "Directional Virtual Coordinate Systems for Wireless Sensor Networks," IEEE International Conference on Communications, 2011, pp. 1-6

[DhJa11c] D. C. Dhanapala and A. P. Jayasumana, "Geo-logical Routing in Wireless Sensor Networks," Proc. of the 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2011, pp. 305-313

[DoKo11] C. Doulaverakis, N. Konstantinou, T. Knape, I. Kompatsiaris, and J, Soldatos, "Sensor Urban Environments Can Prevent and Fight Robberies, Terrorism Attacks, Natural Catastrophes and Other Emergencies," Retrieved April 5, 2013 from: http://www.theamericaspostes.com/?p=3752, Sep. 2011

[EgVa05] E. Egea-López, J. Vales-Alonso, A. Martínez-Sala, P. Pavón-Marñio, and J. García-Haro, "Simulation Tools for Wireless Sensor Networks," Proc. of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems, 2005

[GuWa07] H. Gu, K. Wang, H. Wang, J. Zhang, and C. Wang, "Routing in Hexagonal Wireless Sensor Networks," Proc. of the International Conference on Wireless and Optical Communications Networks, 2007, pp. 1--5

[HeKu99] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks", Proc. of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking, 1999, pp. 174-185

[HeXi06] M. He and W. Xiao, "A Unified Addressing Schema for Hexagonal and Honeycomb Networks with Isomorphic Cayley Graphs," Proc. of the 1st International Multi-Symposiums on Computer and Computational Sciences, 2006, vol. 1, pp. 363-368

[HoLi05] J. C. Hou and N. Li, "Topology Construction and Maintenance in Wireless Sensor Networks," Handbook of Sensor Networks: Algorithms and Architectures, 2005, pp. 311--341

[Hu93] L. Hu, "Topology Control for Multihop Packet Radio Networks," IEEE Transactions on Communications, 1993, vol.41, no.10, pp.1474-1481

[HuHo05] T. Hui, S. Hong, and T. Matsuzawa, "Developing Energy-Efficient Topologies and Routing for Wireless Sensor Networks" Proc. of the 2005 IFIP International Conference on Network and Parallel Computing, 2005, pp. 461-469

[HuHs08] J. Huang, P. Hsu, and K. Chen, "PyraNet: An Efficient and Reliable Pyramidal Wireless Sensor Network," Proc. of the 5th ACM Symposium on Performance Evaluation of Wireless Ad Hoc Sensor, and Ubiquitous Networks, 2008, pp. 50-53

[InGo03] C.Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, “Directed Diffusion for Wireless Sensor Networking”. IEEE/ACM Transactions on Networking, 2003, vol. 11, no.1, pp. 2-16

[JaPu01] R. Jain, A. Puri, and R. Sengupta, "Geographical Routing using Partial Information for Wireless Ad Hoc Networks," IEEE Personal Communications, 2001, vol.8, no.1, pp. 48-57

[JeZo09] M. Jetvic, N. Zogovic, and G. Dimic, “Evaluation of Wireless Sensor Network Simulators,” TELFOR 2009

[KaKu00] B. Karp and H. T. Kung, “GPSR: Greedy Perimeter Stateless Routing for Wireless Networks,” Proc. of the 6th Annual International Conference on Mobile Computing and Networking, 200, pp. 243-254

[KoSa09] M. Korkalainen, M. Sallinen, N. Karkkainen, and P. Tukeva, "Survey of Wireless Sensor Networks Simulation Tools for Demanding Applications," Proc. Of the 5th International Conference on Networking and Services, 2009, pp. 102-106

[KuWa03a] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, “Geometric Ad-Hoc Routing: Of Theory and Practice,” Proc. of the 22nd Annual Symposium on Principles of Distributed Computing, 2003, pp. 63-72

[KuWa03b] F Kuhn, R Wattenhofer, and A. Zollinger, “Worst-Case Optimal And Average-Case Efficient Geometric Ad-Hoc Routing,” Proc. of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2003, pp. 267-278

[LiAb08] K. Liu and N. Abu, “Aligned Virtual Coordinates for Greedy Geometric Routing in WSNs,” International Journal on Sensor Networks, 2008, pp. 252-265

[LiJa00] J. Li, J. Jannotti, S. Douglas, D. R. Karger, and R. Morris, “A Scalable Location Service for Geographic Ad Hoc Routing,” Proc. of the 6th Annual International Conference on Mobile Computing and Networking, 2000, pp. 120-130

[MuAm10] R. Mulligan and H.M. Ammari, “Coverage in Wireless Sensor Networks: A Survey,” Journal of Network Protocols and Algorithms, 2010, vol. 2, no. 2, pp. 27–53

[NoSt02] F. G. Nocetti, I. Stojmenovic, and J. Zhang, “Addressing and Routing in Hexagonal Networks with Applications for Tracking Mobile Users and Connection Rerouting in Cellular Networks,” IEEE Transactions on Parallel Distributed Systems, 2002, pp. 963-971

[Pe97] C. Perkins, “Ad-Hoc On Demand Distance Vector Routing (AODV),” Internet-Draft, 1997

[RaDe12] M. Radi, B. Dezfouli, K. Bakar, and M. Lee, "Multipath Routing in Wireless Sensor Networks: Survey and Research Challenges," Sensors, 2012, vol. 12, no. 1, pp. 650-685

[RaRa03] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and Ion Stoica, “Geographic Routing Without Location Information,” Proc. of the 9th Annual International Conference on Mobile Computing and Networking, 2003, pp. 96-108

[RoMa04] K. Romer and F. Mattern, "The Design Space of Wireless Sensor Networks," IEEE Wireless Communications, 2004, vol.11, no.6, pp. 54- 61

[Sa05] P. Santi, “Topology Control in Wireless Ad Hoc and Sensor Networks,” ACM Computing Surveys, 2005, vol. 37, no.2, pp. 164-194

[SaWe01] A. Salhieh, J. Weinmann, M. Kochhal, and L. Schwiebert, “Power Efficient Topologies for Wireless Sensor Networks,” Proc. of the 2001 International Conference on Parallel Processing, 2001, pp. 156-166

[SeHe04] K. Seada, A. Helmy, and R. Govindan,“On the Effect of Localization Errors on Geographic Face Routing in Sensor Networks,” Proc. of the 3rd International Symposium on Information Processing in Sensor Networks, 2004, pp. 71-80

[ShDi07] J. Sheu, M. Ding, and K. Hsieh, "Routing with Hexagonal Virtual Coordinates in Wireless Sensor Networks," Proc. of the Wireless Communications and Networking Conference, 2007, pp. 2929-2934

[ShQa08] A. Sharieh, M. Qatawneh, W. Almobaideen, and A. Sleit, “Hex-Cell: Modeling, Topological Properties and Routing Algorithm,” European Journal of Scientific Research, 2008, vol. 22, no. 2, pp. 457–468

[ShRa02] R.C. Shah and J.M. Rabaey, "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks," Proc. of the IEEE Wireless Communications and Networking Conference, 2002, vol.1, pp. 350- 355

[ShRu03] Y. Shang, W. Ruml, Y. Zhang, and P. Markus, “Localization from Mere Connectivity,” Proc. of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing, 2003, pp. 201-212

[St97] I. Stojmenović, “Honeycomb Networks: Topological Properties and Communication Algorithms,” IEEE Transactions on Parallel & Distributed Systems, 1997, pp. 1036-1042

[SuLi11] H. Sundani, H. Li, V. Devabhaktuni, M. Alam, and P. Bhattacharya, "Wireless Sensor Network Simulators A Survey and Comparisons". International Journal of Computer Networks, 2011, vol. 2, no.5, pp. 249–265

[TiTe11] Tibbo Technology, “Sensor Networks,” Retrieved April 5, 2013 from: http://aggregate.tibbo.com/solutions/sensor_network.html, Jan. 2011

[TsYa09] M. Tsai, H. Yang, B. Liu, and W. Huang, “Virtual-Coordinate based Delivery-Guaranteed Routing Protocol in Wireless Sensor Networks,” IEEE/ACM Transactions on Networking, 2009, pp. 1228-1241

[Yu11] F. Yu, “A Survey of Wireless Sensor Network Simulation Tools,” In Washington University in St. Louis, Department of Science and Engineering, http://www.cse.wustl.edu/~jain/cse567-11/ftp/sensor/index.html

[YuEs01] Y. Yu, D. Estrin, and R. Govindan, “Geographical and Energy-Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks”. UCLA Computer Science Department Technical Report, 2001, UCLA-CSD TR-01-0023

[WiLa09] P. Wightman and M. A. Labrador, “Topology Maintenance: Extending the Lifetime of Wireless Sensor Networks,” IEEE LantinCom, 2009

[WiLa10] P. Wightman and M. A. Labrador, “Topology Control in Wireless Sensor Networks,” Tutorial given in IEEE Latincom, 2010

[ZhCh10] J. Zhou, Y. Chen, B. Leong, and P. Sundaramoorthy, “Practical 3D Geographic Routing for Wireless Sensor Networks,” Proc. of the 8th ACM Conference on Embedded Networked Sensor Systems, 2010, pp. 337-350

[ZhHo04] H. Zhang and J. Hou, “Maintaining Sensing Coverage and Connectivity in Large Sensor Networks,” NSF International Workshop on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wirelsss, and Peer-to-Peer Networks, 2004

[ZhJi08] F. Zhang, A. Jiang, and J. Chen, “Robust Planarization of Unlocalized Wireless Sensor Networks,” Proc. of the INFOCOM, 2008, pp. 798--806

Appendix A

Source Code

A.1 Main GUI Code

```
%--------------------------------------------------------------------------
%
%     Copyright (C) <2012>  <Pritam Shah, and Anura Jayasumana>
%     Colorado State University, Fort Collins, CO.
%     http://www.cnrl.colostate.edu/
%
%--------------------------------------------------------------------------

function varargout = VCSIMTOOL(varargin)
% VCSIMTOOL M-file for VCSIMTOOL.fig
%      VCSIMTOOL, by itself, creates a new VCSIMTOOL or raises the existing
%      singleton*.
%
%      H = VCSIMTOOL returns the handle to a new VCSIMTOOL or the handle to
%      the existing singleton*.
%
%      VCSIMTOOL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in VCSIMTOOL.M with the given input arguments.
%
%      VCSIMTOOL('Property','Value',...) creates a new VCSIMTOOL or raises 
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before VCSIMTOOL_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to VCSIMTOOL_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help VCSIMTOOL

% Last Modified by GUIDE v2.5 05-Nov-2012 16:12:51

% Begin initialization code - DO NOT EDIT
gui_Singleton = 0;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @VCSIMTOOL_OpeningFcn, ...
    'gui_OutputFcn',  @VCSIMTOOL_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Description: Initializes the GUI with values provided in
% harraysize and varraysize edit box.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --- Executes just before VCSIMTOOL is made visible.
function VCSIMTOOL_OpeningFcn(hObject, ~, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to VCSIMTOOL (see VARARGIN)
handles.hsize = str2double(get(handles.edit1_harraysize,'String'));
% Get the No. of Columns
handles.vsize = str2double(get(handles.edit2_varraysize,'String'));
% Get the No. of Rows
cla;
clc;
global GanchorPos;
% Global variable for anchors - easy for debugging
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global GvoidNo;
global GvoidVertices;
global placement_error1;
global placement_error2;

global adjmatrix1;
adjmatrix1= [];

GanchorPos = [];
GnetworkNodeType = [];
GnetworkNodePos = [];
GvoidNo = 1;
GvoidVertices = [];
handles.range = 1;                                           % Default range
of each node - 1 radio unit

set(handles.edit16_placement_error,'String','0');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define Network
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
GnetworkNodeType = ones(handles.vsize,handles.hsize);                    %
Define network size and node types
[nodeRow nodeCol] = find(GnetworkNodeType==1);
placement_error = str2double(get(handles.edit16_placement_error,'String'));
placement_error = 2*placement_error;
placement_error1 = -(placement_error/2) +
placement_error.*rand(size(nodeRow,1),1);
placement_error1 = round(placement_error1*100)/100;
```

```
placement_error2 = -(placement_error/2) +
placement_error.*rand(size(nodeCol,1),1);
placement_error2 = round(placement_error2*100)/100;

nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;                                          %
Define Node positions

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.rangeBackup = handles.range;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;

updateTotalNodesAnchors( handles );                         % Function
updates the Total Nodes and Total ANchors box in GUI

handles.ah = gca;                                           % Axis handle
stored to retrive the data from axis - UNDO function
set(handles.edit4_nodeRange,'String',handles.range);
set(handles.pushbutton5_setAnchor,'UserData',0);
set(handles.edit7_node_density,'visible','off');            % By default set
node density textbox off
set(handles.text10_nodeDensity,'visible','off');
set(handles.slider2_node_density,'visible','off');
set(handles.pushbutton6_set_node_density,'visible','off');
set(handles.pushbutton1_undo,'visible','off');
% Choose default command line output for VCSIMTOOL
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes VCSIMTOOL wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = VCSIMTOOL_OutputFcn(~, ~, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on selection change in popupmenu1_drawVoid.
function popupmenu1_drawVoid_Callback(hObject, ~, handles)
% hObject    handle to popupmenu1_drawVoid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
popupmenu1_drawVoid contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1_drawVoid
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global GvoidVertices;
global GvoidNo;
global placement_error1;
global placement_error2;
global adjmatrix1;
adjmatrix1= [];

% Backup for undo function
handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.rangeBackup = handles.range;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;

str = get(hObject, 'String');
val = get(hObject, 'Value');
hold on;

switch str{val}
    case 'Linear' % Draw Linear lines between points
        xy = [];
        n = 0;
        % Loop, picking up the points.
        disp('Left mouse button picks points.')
        disp('Right mouse button picks last point.')
        but = 1;
        while but == 1
            [xi,yi,but] = ginput(1);
            plot(xi,yi,'ro')
            n = n+1;
            xy(:,n) = [xi;yi];
        end
        % Plot the interpolated curve.
        mask = poly2mask(xy(1,:),xy(2,:),handles.vsize,handles.hsize);
        mask = ~mask;
        xy = round(xy);
    case 'Interpolation' %Interpolation between selected points
        xy = [];
        n = 0;
        % Loop, picking up the points.
        disp('Left mouse button picks points.')
        disp('Right mouse button picks last point.')
        but = 1;
        while but == 1
            [xi,yi,but] = ginput(1);
            plot(xi,yi,'ro')
            n = n+1;
            xy(:,n) = [xi;yi];
        end
        % Interpolate with a spline curve and finer spacing.
        t = 1:n;
        ts = 1: 0.1: n;
        xys = spline(t,xy,ts);
        % Plot the interpolated curve.
        mask = poly2mask(xys(1,:),xys(2,:),handles.vsize,handles.hsize);
        mask = ~mask;
        xy = round(xys);
    case 'Rectangle' % Rectangle for selected points
        rect = getrect(gcf);
        xy(1,1) = rect(1); xy(2,1) = rect(2);
        xy(1,2) = rect(1); xy(2,2) = rect(2) + rect(4);
        xy(1,3) = rect(1)+rect(3); xy(2,3) = rect(2) + rect(4);
        xy(1,4) = rect(1)+rect(3); xy(2,4) = rect(2);
        xy(:,5) = xy(:,1);
        mask = poly2mask(xy(1,:),xy(2,:),handles.vsize,handles.hsize);
        mask = ~mask;
        xy = round(xy);
    case 'Circle' % Circle for selected points
        circ = getrect(gcf);
        xc = circ(1) + circ(3)/2;
        yc = circ(2) + circ(4)/2;
        xy = calculateEllipse(xc,yc,circ(3)/2,circ(4)/2,0);
        mask = poly2mask(xy(:,1),xy(:,2),handles.vsize,handles.hsize);
        mask = ~mask;
        xy = round(xy');
end

%***********************************
% To store the Voids
%***********************************
xmax = max(xy(1,:));
ymax = max(xy(2,:));
xmin = min(xy(1,:));
ymin = min(xy(2,:));
xi = [xmin xmax];
yi = [ymin ymax];
[p,q] = meshgrid(xi, yi);
GvoidVertices(:,:,GvoidNo) =  [p(:) q(:)];%xy';
GvoidVertices([3 4],:,GvoidNo)= GvoidVertices([4 3],:,GvoidNo);
set(handles.edit18_voidVertices,'String',mat2str(GvoidVertices(:,:,GvoidNo)))
;
GvoidNo = GvoidNo+1;
%***********************************

GnetworkNodeType = GnetworkNodeType .* mask;
```

```
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
matrix_size = size(GnetworkNodeType);
GnetworkNodePos = zeros(matrix_size);
GnetworkNodePos(nodes)= node_ind;
GanchorPos =
GnetworkNodePos(find(ismember(handles.Topology.nodePos,GanchorPos)))';
% Anchor position updated
GanchorPos(GanchorPos==0)=[];
GanchorPos = unique_no_sort(GanchorPos);

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );

% % hold on
% % plot(GvoidVertices(:,1,(GvoidNo-1)),GvoidVertices(:,2,(GvoidNo-1)),'k');
% % hold off
% rectangle('position', [xmin,ymin,(xmax-xmin),(ymax-ymin)]);
set(handles.pushbutton1_undo,'visible','on');
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function popupmenu1_drawVoid_CreateFcn(hObject, ~, ~)
% hObject    handle to popupmenu1_drawVoid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit1_harraysize_Callback(hObject, ~, handles)
% hObject    handle to edit1_harraysize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1_harraysize as text
%        str2double(get(hObject,'String')) returns contents of
edit1_harraysize as a double
handles.hsize = str2double(get(hObject,'String'));
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1= [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos = [];
GvoidVertices = [];
GvoidNo = 1;

%checks to see if input is empty. if so, default harraysize to ten
if (handles.hsize < 10)
    set(hObject,'String','10')
    handles.hsize = 10;
end
GnetworkNodeType = ones(handles.vsize,handles.hsize);                    %
Define network size and node types
[nodeRow nodeCol] = find(GnetworkNodeType==1);
placement_error = str2double(get(handles.edit16_placement_error,'String'));
placement_error = 2*placement_error;
placement_error1 = -(placement_error/2) +
placement_error.*rand(size(nodeRow,1),1);
placement_error1 = round(placement_error1*100)/100;
placement_error2 = -(placement_error/2) +
placement_error.*rand(size(nodeCol,1),1);
placement_error2 = round(placement_error2*100)/100;
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;                                          %
Define Node positions

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );                             % Function
updates the Total Nodes and Total ANchors box in GUI

set(handles.pushbutton1_undo,'visible','on');
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function edit1_harraysize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1_harraysize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_varraysize_Callback(hObject, eventdata, handles)
% hObject    handle to edit2_varraysize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2_varraysize as text
%        str2double(get(hObject,'String')) returns contents of
edit2_varraysize as a double
handles.vsize = str2double(get(hObject,'String'));
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1= [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos = [];
GvoidVertices = [];
GvoidNo = 1;

%checks to see if input is empty. if so, default varraysize to ten
if (handles.vsize < 10)
    set(hObject,'String','10')
    handles.vsize = 10;
end
GnetworkNodeType = ones(handles.vsize,handles.hsize);                  %
Define network size and node types
[nodeRow nodeCol] = find(GnetworkNodeType==1);
placement_error = str2double(get(handles.edit16_placement_error,'String'));
placement_error = 2*placement_error;
placement_error1 = -(placement_error/2) +
placement_error.*rand(size(nodeRow,1),1);
```

```
placement_error1 = round(placement_error1*100)/100;
placement_error2 = -(placement_error/2) +
placement_error.*rand(size(nodeCol,1),1);
placement_error2 = round(placement_error2*100)/100;
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;                                          %
Define Node positions

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );                          % Function
updates the Total Nodes and Total ANchors box in GUI
set(handles.pushbutton1_undo,'visible','on');
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function edit2_varraysize_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2_varraysize (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1_undo.
function pushbutton1_undo_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1_undo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global GvoidVertices;
global GvoidNo;
global placement_error1;
global placement_error2;
global adjmatrix1;
adjmatrix1= [];

% child = get(handles.ah, 'children');
% delete(child(1));

placement_error1 = handles.placement_error1_backup;
placement_error2 = handles.placement_error2_backup;
```

```
handles.Topology = handles.topologyBackup;
GanchorPos = handles.anchorPosBackup;                                    % Restore
the previous anchor_pos matrix
handles.voidVerticesBackup = GvoidVertices;
GvoidNo = handles.voidNoBbackup;
handles.range = handles.rangeBackup;

GnetworkNodeType = handles.Topology.nodeType;
GnetworkNodePos = handles.Topology.nodePos;
[handles.vsize handles.hsize] = size(GnetworkNodeType);
set(handles.edit2_varraysize,'String',handles.vsize);
set(handles.edit1_harraysize,'String',handles.hsize);
updateTotalNodesAnchors( handles );
if(GvoidNo > 1)
    %     GvoidNo = GvoidNo -1;

set(handles.edit18_voidVertices,'String',mat2str(GvoidVertices(:,:,(GvoidNo-
1))));
end
set(handles.pushbutton1_undo,'visible','off');
% handles = rmfield(handles,'backup');
guidata(hObject,handles);


% --- Executes on selection change in popupmenu2_anchorPlacement.
function popupmenu2_anchorPlacement_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2_anchorPlacement (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
popupmenu2_anchorPlacement contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2_anchorPlacement
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;


handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;


str = get(hObject,'String');
```

```
val = get(hObject,'Value');
matrix_size = size(GnetworkNodeType);
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
switch str{val}
    case 'Random'
        set(handles.edit_sliderText,'visible','on');
        set(handles.slider1_anchorDensity,'visible','on');
        set(handles.text5_anchorDensity,'visible','on');
        set(handles.pushbutton2_set_anchorDensity,'visible','on');
        set(handles.text7_anchorXcoordn,'visible','off');
        set(handles.edit5_anchorXcoordn,'visible','off');
        set(handles.text8_anchorYcoordn,'visible','off');
        set(handles.edit6_anchorYcoordn,'visible','off');
        set(handles.pushbutton5_setAnchor,'visible','off');
    case 'Manual (mouse)'
        set(handles.edit_sliderText,'visible','off');
        set(handles.slider1_anchorDensity,'visible','off');
        set(handles.text5_anchorDensity,'visible','off');
        set(handles.pushbutton2_set_anchorDensity,'visible','off');
        set(handles.text7_anchorXcoordn,'visible','off');
        set(handles.edit5_anchorXcoordn,'visible','off');
        set(handles.text8_anchorYcoordn,'visible','off');
        set(handles.edit6_anchorYcoordn,'visible','off');
        set(handles.pushbutton5_setAnchor,'visible','off');
        but = 1;
        hold on
        while but ==1
            [xi,yi,but] = ginput(1);
            xi=round(xi);yi=round(yi);
            if(xi<=handles.hsize && yi<=handles.vsize)
                if(GnetworkNodeType(yi,xi) == 1)
                    linInd = sub2ind([handles.vsize handles.hsize], yi, xi);
                    plot(xi,yi,'r.')
                    GnetworkNodeType(yi,xi) = 2;
                    GanchorPos = cat(2,GanchorPos,
GnetworkNodePos(sub2ind(size(GnetworkNodeType),yi,xi)));
                end
            end
        end
        %          GanchorPos = GnetworkNodePos(GanchorPos);
        GanchorPos = unique_no_sort(GanchorPos);
    case 'Manual (coordinates)'
        set(handles.edit_sliderText,'visible','off');
        set(handles.slider1_anchorDensity,'visible','off');
        set(handles.text5_anchorDensity,'visible','off');
        set(handles.pushbutton2_set_anchorDensity,'visible','off');
        set(handles.text7_anchorXcoordn,'visible','on');
        set(handles.edit5_anchorXcoordn,'visible','on');
        set(handles.text8_anchorYcoordn,'visible','on');
        set(handles.edit6_anchorYcoordn,'visible','on');
        set(handles.pushbutton5_setAnchor,'visible','on');
        waitfor(handles.pushbutton5_setAnchor,'UserData');
    case 'ENS (mouse)'
        set(handles.edit_sliderText,'visible','off');
        set(handles.slider1_anchorDensity,'visible','off');
        set(handles.text5_anchorDensity,'visible','off');
        set(handles.pushbutton2_set_anchorDensity,'visible','off');
        set(handles.text7_anchorXcoordn,'visible','off');
        set(handles.edit5_anchorXcoordn,'visible','off');
        set(handles.text8_anchorYcoordn,'visible','off');
        set(handles.edit6_anchorYcoordn,'visible','off');
        set(handles.pushbutton5_setAnchor,'visible','off');
        but = 1;
        hold on
        while but <=2
            [xi,yi] = ginput(1);
            xi=round(xi);yi=round(yi);
            if(xi<=handles.hsize && yi<=handles.vsize)
                if(GnetworkNodeType(yi,xi) ~= 0)
                    linInd = sub2ind([handles.vsize handles.hsize], yi, xi);
                    plot(xi,yi,'r.')
                    GnetworkNodeType(yi,xi) = 2;
                    but = but +1;
                end
            end
        end
        hold off
        range = handles.range;
        anchors = find(GnetworkNodeType==2);
        sensors = find(GnetworkNodeType==1);
        no_anchors = numel(anchors);
        no_sensors = numel(sensors);
        VC_matrix = zeros(no_anchors,no_sensors+no_anchors);

        [row_cordn col_cordn] = find(GnetworkNodePos);
        r =1;
        if(isempty(adjmatrix1))
            for k = 1:no_nodes
                for i = floor(max(1,row_cordn(k)-
range)):ceil(min(matrix_size(1),(row_cordn(k)+range)))
                    for j = floor(max(1,col_cordn(k)-
range)):ceil(min(matrix_size(2),(col_cordn(k)+range)))
                        nodeIndex1 = sub2ind(size(GnetworkNodeType), i, j);
                        nodeIndex2 = sub2ind(size(GnetworkNodeType),
row_cordn(k), col_cordn(k));
                        if (GnetworkNodeType(i,j) ~= 0)
                            if (i ~= row_cordn(k) || j ~= col_cordn(k))
                                if(sqrt((abs(i+placement_error1(nodeIndex1)-
row_cordn(k)-placement_error1(nodeIndex2))^2 +
abs(j+placement_error2(nodeIndex1)-col_cordn(k)-
placement_error2(nodeIndex2))^2)) <= (handles.range))
                                    pairs(r,1) =
GnetworkNodePos(row_cordn(k),col_cordn(k));
                                    pairs(r,2) = GnetworkNodePos(i,j);
                                    r = r + 1;
                                end
                            end
                        end
                    end
                end
            end
            adjmatrix1 = accumarray(pairs, 1);
            if(size(adjmatrix1,1) < no_nodes)
                adjmatrix1(no_nodes,1) = 0;
                adjmatrix1(1,no_nodes) = 0;
            end
        end
        GanchorPos = GnetworkNodePos(anchors);
        GanchorPos = unique_no_sort(GanchorPos);
        S = sparse(adjmatrix1);
        for i = 1:no_anchors
            VC_matrix(i,:)=dijkstra_sp(S,GanchorPos(i));
        end
        DVC_function = (VC_matrix(1,:).^2-
VC_matrix(2,:).^2)./(2*VC_matrix(2,GanchorPos(1)));
        for i = 1:no_nodes
            if(i ~= GanchorPos(1) || i ~= GanchorPos(2))
                node_neighbors = [];
                node_neighbors = find(adjmatrix1(i,:));
                if ((isempty(node_neighbors)==0)&&((DVC_function(i) <
min(DVC_function(node_neighbors)))  || (DVC_function(i) >
max(DVC_function(node_neighbors))) ))
                    selected_anchors(i) = i;
                end
            end
        end
        selected_anchors(selected_anchors == 0) = [];
        linear_anchor_ind =
sub2ind(size(GnetworkNodeType),row_cordn(selected_anchors),col_cordn(selected
_anchors));
        GnetworkNodeType(linear_anchor_ind) = 2;
        GanchorPos = cat(2,GanchorPos, selected_anchors);
        GanchorPos = unique_no_sort(GanchorPos);
        GanchorPos = unique_no_sort(GanchorPos);
    case 'ENS (coordinates)'
        set(handles.edit_sliderText,'visible','off');
        set(handles.slider1_anchorDensity,'visible','off');
        set(handles.text5_anchorDensity,'visible','off');
        set(handles.pushbutton2_set_anchorDensity,'visible','off');
        set(handles.text7_anchorXcoordn,'visible','on');
        set(handles.edit5_anchorXcoordn,'visible','on');
        set(handles.text8_anchorYcoordn,'visible','on');
        set(handles.edit6_anchorYcoordn,'visible','on');
        set(handles.pushbutton5_setAnchor,'visible','on');
        waitfor(handles.pushbutton5_setAnchor,'UserData');
        temp = get(handles.pushbutton5_setAnchor,'UserData');
        GnetworkNodeType(temp) =2;
        waitfor(handles.pushbutton5_setAnchor,'UserData');
        temp = get(handles.pushbutton5_setAnchor,'UserData');
        set(handles.pushbutton5_setAnchor,'UserData',0);
        GnetworkNodeType(temp) =2;
        range = handles.range;
        anchors = find(GnetworkNodeType==2);
        sensors = find(GnetworkNodeType==1);
        no_anchors = numel(anchors);
        no_sensors = numel(sensors);
        VC_matrix = zeros(no_anchors,no_sensors+no_anchors);
        [row_cordn col_cordn] = find(GnetworkNodePos);
        r =1;
        if(isempty(adjmatrix1))
            for k = 1:no_nodes
                for i = floor(max(1,row_cordn(k)-
range)):ceil(min(matrix_size(1),(row_cordn(k)+range)))
                    for j = floor(max(1,col_cordn(k)-
range)):ceil(min(matrix_size(2),(col_cordn(k)+range)))
                        nodeIndex1 = sub2ind(size(GnetworkNodeType), i, j);
                        nodeIndex2 = sub2ind(size(GnetworkNodeType),
row_cordn(k), col_cordn(k));
                        if (GnetworkNodeType(i,j) ~= 0)
                            if (i ~= row_cordn(k) || j ~= col_cordn(k))
                                if(sqrt((abs(i+placement_error1(nodeIndex1)-
row_cordn(k)-placement_error1(nodeIndex2))^2 +
abs(j+placement_error2(nodeIndex1)-col_cordn(k)-
placement_error2(nodeIndex2))^2)) <= (handles.range))
                                    pairs(r,1) =
GnetworkNodePos(row_cordn(k),col_cordn(k));
                                    pairs(r,2) = GnetworkNodePos(i,j);
                                    r = r + 1;
                                end
                            end
                        end
                    end
                end
            end
            adjmatrix1 = accumarray(pairs, 1);
            if(size(adjmatrix1,1) < no_nodes)
                adjmatrix1(no_nodes,1) = 0;
                adjmatrix1(1,no_nodes) = 0;
            end
        end
        GanchorPos = GnetworkNodePos(anchors);
        GanchorPos=unique_no_sort(GanchorPos);
        S = sparse(adjmatrix1);
        for i = 1:no_anchors
            VC_matrix(i,:)=dijkstra_sp(S,GanchorPos(i));
        end
        DVC_function = (VC_matrix(1,:).^2-
VC_matrix(2,:).^2)./(2*VC_matrix(2,GanchorPos(1)));
        for i = 1:no_nodes
            if(i ~= GanchorPos(1) || i ~= GanchorPos(2))
                node_neighbors = [];
                node_neighbors = find(adjmatrix1(i,:));
                if ((isempty(node_neighbors)==0)&&((DVC_function(i) <
min(DVC_function(node_neighbors)))  || (DVC_function(i) >
max(DVC_function(node_neighbors))) ))
                    selected_anchors(i) = i;
                end
            end
        end
        selected_anchors(selected_anchors == 0) = [];
        linear_anchor_ind =
sub2ind(size(GnetworkNodeType),row_cordn(selected_anchors),col_cordn(selected
_anchors));
        GnetworkNodeType(linear_anchor_ind) = 2;
        GanchorPos = cat(2,GanchorPos, selected_anchors);
        GanchorPos = unique_no_sort(GanchorPos);
end
handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );                          % Function
updates the Total Nodes and Total ANchors box in GUI
% Update handles structure
set(handles.pushbutton1_undo,'visible','on');
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function popupmenu2_anchorPlacement_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2_anchorPlacement (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider1_anchorDensity_Callback(hObject, eventdata, handles)
% hObject    handle to slider1_anchorDensity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

%obtains the slider value from the slider component
sliderValue = get(handles.slider1_anchorDensity,'Value');

%puts the slider value into the edit text component
set(handles.edit_sliderText,'String', num2str(sliderValue));

% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function slider1_anchorDensity_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1_anchorDensity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function edit_sliderText_Callback(hObject, eventdata, handles)
% hObject    handle to edit_sliderText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_sliderText as text
%        str2double(get(hObject,'String')) returns contents of
edit_sliderText as a double

%get the string for the editText component
sliderValue = get(handles.edit_sliderText,'String');

%convert from string to number if possible, otherwise returns empty
sliderValue = str2double(sliderValue);

%if user inputs something is not a number, or if the input is less than 0
%or greater than 100, then the slider value defaults to 0
if (isempty(sliderValue) || sliderValue < 0 || sliderValue > 100)
    set(handles.slider1_anchorDensity,'Value',0);
    set(handles.edit_sliderText,'String','0');
else
    set(handles.slider1_anchorDensity,'Value',sliderValue);
end

% --- Executes during object creation, after setting all properties.
function edit_sliderText_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_sliderText (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in pushbutton2_set_anchorDensity.
function pushbutton2_set_anchorDensity_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2_set_anchorDensity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos = [];

anchors = str2double(get(handles.edit_sliderText,'String'));
nodes = nnz(GnetworkNodeType);
anchors = ceil((anchors/100)*nodes);
size_topology = numel(GnetworkNodeType);
rand_anchors = [ones(anchors,1);zeros((size_topology-anchors),1)];
idx_1 = find(GnetworkNodeType);
idx_0 = find(GnetworkNodeType==0);
rand_seq = randperm(numel(idx_1));
A = idx_1(rand_seq);
A = [A;idx_0];
A = sparse(A);
for i = 1:anchors
    B(A(i)) = rand_anchors(i);
end
B = [B zeros(1,(size_topology - numel(B)))];
B = reshape(B,[handles.vsize handles.hsize]);
GnetworkNodeType = GnetworkNodeType + B;
temp = GnetworkNodePos(GnetworkNodeType==2);
GanchorPos = cat(2,GanchorPos,temp');
GanchorPos = unique_no_sort(GanchorPos);

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );                          % Function
updates the Total Nodes and Total ANchors box in GUI
% Update handles structure
set(handles.pushbutton1_undo,'visible','on');
guidata(hObject, handles);


% --- Executes on button press in pushbutton3_start.
function pushbutton3_start_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3_start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error2;
global placement_error1;
global adjmatrix1;
```

```
range = handles.range;
sensors = find(GnetworkNodeType==1);
no_sensors = numel(sensors);
matrix_size = size(GnetworkNodeType);
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
no_anchors = numel(GanchorPos);
global VC_matrix;
VC_matrix = zeros(no_anchors,no_sensors+no_anchors);
[row_cordn col_cordn] = find(GnetworkNodePos);
r =1;
if(isempty(adjmatrix1))
    for k = 1:no_nodes
        for i = floor(max(1,row_cordn(k)-
range)):ceil(min(matrix_size(1),(row_cordn(k)+range)))
            for j = floor(max(1,col_cordn(k)-
range)):ceil(min(matrix_size(2),(col_cordn(k)+range)))
                nodeIndex1 = sub2ind(size(GnetworkNodeType), i, j);
                nodeIndex2 = sub2ind(size(GnetworkNodeType), row_cordn(k),
col_cordn(k));
                if (GnetworkNodeType(i,j) ~= 0)
                    if (i ~= row_cordn(k) || j ~= col_cordn(k))
                        if(sqrt((abs(i+placement_error1(nodeIndex1)-
row_cordn(k)-placement_error1(nodeIndex2))^2 +
abs(j+placement_error2(nodeIndex1)-col_cordn(k)-
placement_error2(nodeIndex2))^2)) <= (handles.range))
                            Gpairs(r,1) =
GnetworkNodePos(row_cordn(k),col_cordn(k));
                            Gpairs(r,2) = GnetworkNodePos(i,j);
                            r = r + 1;
                        end
                    end
                end
            end
        end
    end
    adjmatrix1 = accumarray(Gpairs, 1);
    if(size(adjmatrix1,1) < no_nodes)
        adjmatrix1(no_nodes,1) = 0;
        adjmatrix1(1,no_nodes) = 0;
    end
end


S_adj = sparse(adjmatrix1);
for i = 1:no_anchors
    VC_matrix(i,:)=dijkstra_sp(S_adj,GanchorPos(i));
end

% *******************  SVD  ***********************************
if(get(handles.checkbox1_svd,'value')==1)
    % Subtracting mean
    %     for ii = 1:no_anchors
    %         temp(ii,:) = VC_matrix(ii,:)-mean(VC_matrix(ii,:));
    %     end
    %     x = unique(VC_matrix','rows');
    %     d = 2;
    %     [U S V]=svd(temp');
    %     P_svd=U(:, 1:d) * S(1:d, 1:d);
    %**********************************
    % As proposed in paper
    [U S V] = svd(VC_matrix');
    P_svd = U*S;
    %**************
    figure
    if(no_anchors > 2)
        plot(P_svd(:,2),P_svd(:,3),'.')
        xlabel(texlabel('P^(2)_SVD'))
        ylabel(texlabel('P^(3)_SVD'))
    else
        plot(P_svd(:,1),P_svd(:,2),'.')
        xlabel(texlabel('P^(1)_SVD'))
        ylabel(texlabel('P^(2)_SVD'))
    end
    title('Reconstructed positions from connectivity information - SVD');
end
% ************************************************************

% ******************  MDS  **********************************
if(get(handles.checkbox2_mds,'value')==1)
    for i = 1:no_nodes
        full_matrix(i,:)=dijkstra_sp(S_adj,i);
    end
    x = unique(full_matrix,'rows');
    D2 = full_matrix.^2;
    d = 2;
    L = eye(no_nodes) - 1/no_nodes * ones(no_nodes);
    temp = -1/2 * L * D2 * L;
    [U, S, V] = svd(temp);
    coord_est = U(:, 1:d) * S(1:d, 1:d).^.5;
    figure;
    plot(-coord_est(:,1), coord_est(:,2), '.');
    %     [U S V]=svd(VC_matrix');
    %     P_svd=U*S;
    %     figure
    %     if(no_anchors > 2)
    %         plot(P_svd(:,2),P_svd(:,3),'.')
    %         xlabel(texlabel('P^(2)_SVD'))
    %         ylabel(texlabel('P^(3)_SVD'))
    %     else
    %         plot(P_svd(:,1),P_svd(:,2),'.')
    %         xlabel(texlabel('P^(1)_SVD'))
    %         ylabel(texlabel('P^(2)_SVD'))
    %     end
    title('Reconstructed positions from connectivity information - MDS');
end
% ************************************************************

anchor_pair_count = 1;
if(no_anchors >=2)
    C = combnk(1:no_anchors,2);
    C = unique(C,'rows');
    anchor_pair_count = size(C,1);
end
DVC_function = zeros(anchor_pair_count,no_nodes);
for ii = 1:anchor_pair_count
    DVC_function(ii,:) = (VC_matrix(C(ii,1),:).^2-
VC_matrix(C(ii,2),:).^2)./(2*VC_matrix(C(ii,1),GanchorPos(C(ii,2))));
end

% *******************  DVC  ***********************************
if(get(handles.checkbox3_dvc,'value')==1)
    DVC_function_t = DVC_function';
    angle_ij = zeros(anchor_pair_count);
    for ii = 1:anchor_pair_count
        for jj = ii+1:anchor_pair_count
            angle_ij(ii,jj)=
acosd(dot(DVC_function_t(:,ii)',DVC_function_t(:,jj))/(norm(DVC_function_t(:,
ii))*norm(DVC_function_t(:,jj))));
        end
    end
    angle_ij = angle_ij + tril(Inf*ones(size(angle_ij)));
    temp_min = min(min(mod(angle_ij,90)));
    [ii jj] = find(mod(angle_ij,90)==temp_min);
    figure;
    plot(DVC_function_t(:,ii), DVC_function_t(:,jj), '.');
    title('Reconstructed positions from connectivity information - DVC');
end
% ************************************************************

% % % %************************************************************
% % % %   Boundary Detect
% % % %************************************************************
% % % figure
% % % plot(P_svd(:,1),P_svd(:,2),'.')
% % % NoNodes=length(P_svd);
% % % hold on
% % % for n=1:NoNodes
% % %     current=P_svd(n,:);i=0;
% % %     for m=1:NoNodes
% % %         if adjmatrix1(m,n)==1
% % %             i=i+1;
% % %             neigh(i,:)=P_svd(m,:);
% % %         end
% % %     end
% % %     if i<=2
% % %                 plot(P_svd(n,1),P_svd(n,2),'k.')
% % % %                 plot(P_svd(n,1),P_svd(n,2),'k*')
% % %     elseif i==3
% % %         flag=triangulararea(neigh,current);
% % %         if flag==1
% % %             plot(P_svd(n,1),P_svd(n,2),'k.')
% % % %             plot(P_svd(n,1),P_svd(n,2),'k*')
% % %         end
% % %     end
% % %     clear neigh;
% % % end
% % % %************************************************************
```

```
gap = [];
figure(handles.figure1);
for ii = 1:no_nodes
    neighbors = find(adjmatrix1(:,ii));
    neighbors_matrix(ii,1:length(neighbors)) = neighbors;
    gap(ii,1:length(neighbors)) = abs(DVC_function(1,ii) -
DVC_function(1,neighbors));
end
gap_round = round(gap*1000)/1000;
global dvc_gap;
dvc_gap = DVC_function';
% dvc_gap(1:no_nodes,anchor_pair_count+1:size(gap_round,2)+anchor_pair_count)
= gap_round;
dvc_gap = round(dvc_gap*1000)/1000;
global unique_gap;
unique_gap = unique(gap_round);
temp = length(unique_gap);
temp = nextpow2(temp);
no_colors = pow2(temp);
color_gap = hsv(no_colors*2);
for i = 1:length(unique_gap)
    unique_gap(i,2:4) = color_gap(i*2-1,:);
end
r = 1;
updateTotalNodesAnchors( handles );
hold on;
Gpairs = [];
[Gpairs(:,1) Gpairs(:,2)] = find(adjmatrix1);
u_pairs = unique(Gpairs,'rows');

r=1;
for i = 1:length(u_pairs)
    if(i>1 && u_pairs(i,1)>u_pairs(i-1,1))
        r = 1;
    end
    temp_gap = gap_round(u_pairs(i,1),r);
    r = r+1;
    temp = find(unique_gap(:,1)==temp_gap);
    nodeIndex1 = sub2ind(size(GnetworkNodeType), row_cordn(u_pairs(i,1)),
col_cordn(u_pairs(i,1)));
    nodeIndex2 = sub2ind(size(GnetworkNodeType), row_cordn(u_pairs(i,2)),
col_cordn(u_pairs(i,2)));
    if(unique_gap(temp,1)<1)%(temp == 1 || temp==5 || temp ==10 || temp ==15
|| temp==20)% || temp ==25 || temp ==30 || temp==35 || temp ==40 || temp==45)
        plot([col_cordn(u_pairs(i,1))+placement_error2(nodeIndex1)
col_cordn(u_pairs(i,2))+placement_error2(nodeIndex2)],[row_cordn(u_pairs(i,1)
)+placement_error1(nodeIndex1)...

row_cordn(u_pairs(i,2))+placement_error1(nodeIndex2)],'color',unique_gap(temp
,2:4));
    end
    if( unique_gap(temp,1)>1)%temp ==25 || temp ==30 || temp==35 || temp ==40
|| temp==45)
```

```
        plot([col_cordn(u_pairs(i,1))+placement_error2(nodeIndex1)
col_cordn(u_pairs(i,2))+placement_error2(nodeIndex2)],[row_cordn(u_pairs(i,1)
)+placement_error1(nodeIndex1)...

row_cordn(u_pairs(i,2))+placement_error1(nodeIndex2)],'color',unique_gap(temp
,2:4),'LineWidth',3);
    end
    if( unique_gap(temp,1)==1)%temp ==25 || temp ==30 || temp==35 || temp
==40 || temp==45)
        plot([col_cordn(u_pairs(i,1))+placement_error2(nodeIndex1)
col_cordn(u_pairs(i,2))+placement_error2(nodeIndex2)],[row_cordn(u_pairs(i,1)
)+placement_error1(nodeIndex1)...

row_cordn(u_pairs(i,2))+placement_error1(nodeIndex2)],'color',unique_gap(temp
,2:4),'LineStyle',':','LineWidth',3);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% To emphasize nodes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[nodeRow nodeCol] = find(handles.Topology.nodeType==1);
oneInd = find(handles.Topology.nodeType==1);
nodeCol = nodeCol + placement_error2(oneInd);
nodeRow = nodeRow + placement_error1(oneInd);
plot(nodeCol,nodeRow,'k.');
set(handles.edit8_numofnodes,'String',size(nodeCol,1));

[nodeRow nodeCol] = find(handles.Topology.nodeType==2);
twoInd = find(handles.Topology.nodeType==2);
nodeCol = nodeCol + placement_error2(twoInd);
nodeRow = nodeRow + placement_error1(twoInd);
hold on
plot(nodeCol,nodeRow,'r.');
set(handles.edit10_numofanchors,'String',size(nodeCol,1));
hold off
axis([0.5 handles.hsize+1 0.5 handles.vsize+1]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

hold off
% % % saveas(gcf,'image.png');
% save(sprintf('vc_matrix%01d',range),'VC_matrix');




function edit4_nodeRange_Callback(hObject, eventdata, handles)
% hObject    handle to edit4_nodeRange (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4_nodeRange as text
%        str2double(get(hObject,'String')) returns contents of
edit4_nodeRange as a double
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1 = [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

handles.range = str2double(get(hObject,'String'));
%checks to see if input is empty. if so, default range to one
if (handles.range < 1)
    set(hObject,'String','1')
    handles.range = 1;
end
set(handles.pushbutton1_undo,'visible','on');
updateTotalNodesAnchors( handles );
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function edit4_nodeRange_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4_nodeRange (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_anchorXcoordn_Callback(hObject, eventdata, handles)
% hObject    handle to edit5_anchorXcoordn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5_anchorXcoordn as
text
%        str2double(get(hObject,'String')) returns contents of
edit5_anchorXcoordn as a double


% --- Executes during object creation, after setting all properties.
function edit5_anchorXcoordn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5_anchorXcoordn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_anchorYcoordn_Callback(hObject, eventdata, handles)
% hObject    handle to edit6_anchorYcoordn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6_anchorYcoordn as
text
%        str2double(get(hObject,'String')) returns contents of
edit6_anchorYcoordn as a double


% --- Executes during object creation, after setting all properties.
function edit6_anchorYcoordn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6_anchorYcoordn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton5_setAnchor.
function pushbutton5_setAnchor_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5_setAnchor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

xcordn = str2double(get(handles.edit5_anchorXcoordn,'String'));
ycordn = str2double(get(handles.edit6_anchorYcoordn,'String'));
GnetworkNodeType(ycordn,xcordn) = 2;
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;                                          %
Define Node positions
temp = sub2ind(size(GnetworkNodeType),ycordn,xcordn);
GanchorPos = cat(2,GanchorPos,GnetworkNodePos(temp));
GanchorPos = unique_no_sort(GanchorPos);

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );

set(handles.pushbutton1_undo,'visible','on');
set(handles.pushbutton5_setAnchor,'UserData',temp);
guidata(hObject, handles);


% --- Executes on button press in togglebutton2_gapdata.
function togglebutton2_gapdata_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2_gapdata (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2_gapdata
button_state = get(hObject,'Value');
if button_state ==1
    dcm_obj = datacursormode(gcf);
    set(dcm_obj,'SnapToDataVertex','off','Enable','on');
    set(dcm_obj,'UpdateFcn',{@myupdategapfcn,dcm_obj});
else
    datacursormode off
end


% --- Executes on button press in togglebutton3_nodedata.
function togglebutton3_nodedata_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton3_nodedata (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton3_nodedata
button_state = get(hObject,'Value');
if button_state ==1
    dcm_obj = datacursormode(gcf);
    set(dcm_obj,'SnapToDataVertex','on','Enable','on');
set(dcm_obj,'UpdateFcn',{@myupdatenodefcn,size(handles.Topology.nodeType)});
else
    datacursormode off
end


% --- Executes on slider movement.
function slider2_node_density_Callback(hObject, eventdata, handles)
% hObject    handle to slider2_node_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider
%obtains the slider value from the slider component
sliderValue = get(handles.slider2_node_density,'Value');

%puts the slider value into the edit text component
set(handles.edit7_node_density,'String', num2str(sliderValue,4));

% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function slider2_node_density_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2_node_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function edit7_node_density_Callback(hObject, eventdata, handles)
% hObject    handle to edit7_node_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7_node_density as text
%        str2double(get(hObject,'String')) returns contents of
edit7_node_density as a double
%get the string for the editText component
sliderValue = get(handles.edit7_node_density,'String');

%convert from string to number if possible, otherwise returns empty
sliderValue = round(str2double(sliderValue)*100)/100;
%if user inputs something is not a number, or if the input is less than 0
```

```
%or greater than 100, then the slider value defaults to 0
if (isempty(sliderValue) || sliderValue < 0 || sliderValue > 100)
    set(handles.slider1_anchorDensity,'Value',0);
    set(handles.edit_sliderText,'String','0');
else
    set(handles.slider1_anchorDensity,'Value',sliderValue);
end
% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function edit7_node_density_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7_node_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton6_set_node_density.
function pushbutton6_set_node_density_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6_set_node_density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1 = [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos=[];
GvoidVertices = [];
GvoidNo = 1;

node_density = get(handles.edit7_node_density,'String');
node_density = str2double(node_density);
```

```
max_no_nodes = handles.vsize*handles.hsize;
node_density = node_density/100;
no_nodes = floor(max_no_nodes*node_density);
node_pos = randi(max_no_nodes,1,no_nodes);
node_pos = unique(node_pos);
while(numel(node_pos)<no_nodes)
    temp = randi(max_no_nodes,1,no_nodes);
    node_pos = cat(2,node_pos,temp);
    node_pos = unique_no_sort(node_pos);
    if(numel(node_pos)>no_nodes)
        node_pos(:,(no_nodes+1:end)) = [];
    end
end
GnetworkNodeType = zeros(handles.vsize,handles.hsize);
GnetworkNodeType(node_pos) = 1;
nodes = find(GnetworkNodeType);
nodeRow = (1:numel(GnetworkNodeType))';
nodeCol = (1:numel(GnetworkNodeType))';
placement_error = str2double(get(handles.edit16_placement_error,'String'));
placement_error = 2*placement_error;
placement_error1 = -(placement_error/2) +
placement_error.*rand(size(nodeRow,1),1);
placement_error1 = round(placement_error1*100)/100;
placement_error2 = -(placement_error/2) +
placement_error.*rand(size(nodeCol,1),1);
placement_error2 = round(placement_error2*100)/100;
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );

set(handles.pushbutton1_undo,'visible','on');
% Update handles structure
guidata(hObject, handles);


% --------------------------------------------------------------------
function import_locations_Callback(hObject, eventdata, handles)
% hObject    handle to import_locations (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1 = [];
```

```
handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos=[];
GvoidVertices = [];
GvoidNo = 1;

[FileName,PathName] = uigetfile('*.mat','Select the MAT-file');
uint32 filedata;
filedata = importdata(fullfile(PathName, FileName));
while (size(filedata,2) ~= 2)
    h = msgbox('Select x-y coordinate file','Error Loading Location File',
'error');
    uiwait(h);
    [FileName,PathName] = uigetfile('*.mat','Select the MAT-file');
    filedata = importdata(fullfile(PathName, FileName));
end
handles.hsize = ceil(max(filedata(:,1)));
handles.vsize = ceil(max(filedata(:,2)));
set(handles.edit1_harraysize,'String',handles.hsize);
set(handles.edit2_varraysize,'String',handles.vsize);
GnetworkNodeType = zeros(handles.vsize, handles.hsize);
linearInd = sub2ind(size(GnetworkNodeType), filedata(:,2),filedata(:,1));
GnetworkNodeType(linearInd) = 1;
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );

set(handles.pushbutton1_undo,'visible','on');
% Update handles structure
guidata(hObject, handles);



% --------------------------------------------------------------------
function import_network_Callback(hObject, eventdata, handles)
% hObject    handle to import_network (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1 = [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos=[];
GvoidVertices = [];
GvoidNo = 1;

[FileName,PathName] = uigetfile('*.mat','Select the MAT-file');
uint32 filedata;
filedata = importdata(fullfile(PathName, FileName));
handles.hsize = size(filedata,2);
handles.vsize = size(filedata,1);
GnetworkNodeType = ones(handles.vsize,handles.hsize);                    %
Define network size and node types
[nodeRow nodeCol] = find(GnetworkNodeType==1);
placement_error = str2double(get(handles.edit16_placement_error,'String'));
placement_error = 2*placement_error;
placement_error1 = -(placement_error/2) +
placement_error.*rand(size(nodeRow,1),1);
placement_error1 = round(placement_error1*100)/100;
placement_error2 = -(placement_error/2) +
placement_error.*rand(size(nodeCol,1),1);
placement_error2 = round(placement_error2*100)/100;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
GnetworkNodeType = filedata;
set(handles.edit1_harraysize,'String',handles.hsize);
set(handles.edit2_varraysize,'String',handles.vsize);
GanchorPos = find(GnetworkNodeType==2);
GanchorPos = unique_no_sort(GanchorPos);
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );

set(handles.pushbutton1_undo,'visible','on');
% Update handles structure
guidata(hObject, handles);


% --------------------------------------------------------------------
```

```
function menu_exit_Callback(hObject, eventdata, handles)
% hObject    handle to menu_exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
delete(handles.figure1);


% --------------------------------------------------------------------
function help_contents_Callback(hObject, eventdata, handles)
% hObject    handle to help_contents (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function about_simulator_Callback(hObject, eventdata, handles)
% hObject    handle to about_simulator (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% out = dialog('WindowStyle', 'normal','resize','on', 'Name', 'About
VCSIMTOOL');
About_VCSIMTOOL();




function edit8_numofnodes_Callback(hObject, eventdata, handles)
% hObject    handle to edit8_numofnodes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8_numofnodes as text
%        str2double(get(hObject,'String')) returns contents of
edit8_numofnodes as a double


% --- Executes during object creation, after setting all properties.
function edit8_numofnodes_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8_numofnodes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a
double


% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function edit10_numofanchors_Callback(hObject, eventdata, handles)
% hObject    handle to edit10_numofanchors (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10_numofanchors as
text
%        str2double(get(hObject,'String')) returns contents of
edit10_numofanchors as a double


% --- Executes during object creation, after setting all properties.
function edit10_numofanchors_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10_numofanchors (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes when selected object is changed in uipanel7_nodedistribution.
function uipanel7_nodedistribution_SelectionChangeFcn(hObject, eventdata,
handles)
% hObject    handle to the selected object in uipanel7_nodedistribution
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was
selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1 = [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos=[];
GvoidVertices = [];
GvoidNo = 1;

switch get(hObject,'String')
    case 'Uniform'
        set(handles.edit7_node_density,'visible','off');
        set(handles.text10_nodeDensity,'visible','off');
        set(handles.slider2_node_density,'visible','off');
        set(handles.pushbutton6_set_node_density,'visible','off');
        GnetworkNodeType = ones(handles.vsize,handles.hsize);
        [nodeRow nodeCol] = find(GnetworkNodeType==1);
        nodes = find(GnetworkNodeType);
        placement_error =
str2double(get(handles.edit16_placement_error,'String'));
        placement_error = 2*placement_error;
        placement_error1 = -(placement_error/2) +
placement_error.*rand(size(nodeRow,1),1);
        placement_error1 = round(placement_error1*100)/100;
        placement_error2 = -(placement_error/2) +
placement_error.*rand(size(nodeCol,1),1);
        placement_error2 = round(placement_error2*100)/100;
        no_nodes = numel(nodes);
        node_ind = 1:no_nodes;
        GnetworkNodePos = zeros(size(GnetworkNodeType));
        GnetworkNodePos(nodes)= node_ind;
        handles.Topology.nodeType = GnetworkNodeType;
        handles.Topology.nodePos = GnetworkNodePos;
        updateTotalNodesAnchors( handles );
    case 'Random'
        set(handles.edit7_node_density,'visible','on');
        set(handles.text10_nodeDensity,'visible','on');
        set(handles.slider2_node_density,'visible','on');
        set(handles.pushbutton6_set_node_density,'visible','on');
end

set(handles.pushbutton1_undo,'visible','on');
% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function uipanel7_nodedistribution_CreateFcn(hObject, eventdata, handles)
% hObject    handle to uipanel7_nodedistribution (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --------------------------------------------------------------------
function export_VCs_Callback(hObject, eventdata, handles)
% hObject    handle to export_VCs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error2;
global placement_error1;
global adjmatrix1;
global VC_matrix;

range = handles.range;
matrix_size = size(GnetworkNodeType);
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
no_anchors = numel(GanchorPos);
[row_cordn col_cordn] = find(GnetworkNodePos);
VC_matrix = zeros(no_anchors,no_nodes);
r =1;
if(isempty(adjmatrix1))
    for k = 1:no_nodes
        for i = floor(max(1,row_cordn(k)-
range)):ceil(min(matrix_size(1),(row_cordn(k)+range)))
            for j = floor(max(1,col_cordn(k)-
range)):ceil(min(matrix_size(2),(col_cordn(k)+range)))
                nodeIndex1 = sub2ind(size(GnetworkNodeType), i, j);
                nodeIndex2 = sub2ind(size(GnetworkNodeType), row_cordn(k),
col_cordn(k));
                if (GnetworkNodeType(i,j) ~= 0)
                    if (i ~= row_cordn(k) || j ~= col_cordn(k))
                        if(sqrt((abs(i+placement_error1(nodeIndex1)-
row_cordn(k)-placement_error1(nodeIndex2))^2 +
abs(j+placement_error2(nodeIndex1)-col_cordn(k)-
placement_error2(nodeIndex2))^2)) <= (handles.range))
                            Gpairs(r,1) =
GnetworkNodePos(row_cordn(k),col_cordn(k));
                            Gpairs(r,2) = GnetworkNodePos(i,j);
                            r = r + 1;
                        end
                    end
                end
            end
        end
    end
    adjmatrix1 = accumarray(Gpairs, 1);
    if(size(adjmatrix1,1) < no_nodes)
        adjmatrix1(no_nodes,1) = 0;
        adjmatrix1(1,no_nodes) = 0;
    end
end


S_adj = sparse(adjmatrix1);
for i = 1:no_anchors
    VC_matrix(i,:)=dijkstra_sp(S_adj,GanchorPos(i));
end
x = [GanchorPos' VC_matrix];
if(no_anchors == 0 )
    h = msgbox('Select x-y coordinate file','Error Loading Location File',
'error');
    uiwait(h);
else
    uisave('x','VirtCoords.mat');
end


% --------------------------------------------------------------------
function export_AM_Callback(hObject, eventdata, handles)
% hObject    handle to export_AM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global adjmatrix1;
uisave('adjmatrix1','Adj_Matrix.mat');



function edit11_source_actual_Callback(hObject, eventdata, handles)
% hObject    handle to edit11_source_actual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11_source_actual as
text
%        str2double(get(hObject,'String')) returns contents of
edit11_source_actual as a double


% --- Executes during object creation, after setting all properties.
function edit11_source_actual_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11_source_actual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit12_vc_source_Callback(hObject, eventdata, handles)
% hObject    handle to edit12_vc_source (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12_vc_source as text
%        str2double(get(hObject,'String')) returns contents of
edit12_vc_source as a double


% --- Executes during object creation, after setting all properties.
function edit12_vc_source_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12_vc_source (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu4_routingmethod.
function popupmenu4_routingmethod_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4_routingmethod (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
popupmenu4_routingmethod contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu4_routingmethod
str = get(hObject, 'String');
val = get(hObject, 'Value');
global GanchorPos;
global GnetworkNodeType;
global GnetworkNodePos;
global adjmatrix1;
global n_in;
global placement_error1;
global placement_error2;
global adjmatrix1;
flag = 0;
global route_hop;
global self;
global VC_matrix;
self = [];
route_hop = 0;

updateTotalNodesAnchors( handles );
hold on;
disp('Select Source');
[xi,yi,but] = ginput(1);
xi=round(xi);yi=round(yi);
if(xi<=handles.hsize && yi<=handles.vsize)
    if(GnetworkNodeType(yi,xi) == 1)
        linInd = sub2ind([handles.vsize handles.hsize], yi, xi);
        plot(xi+placement_error2(linInd),yi+placement_error1(linInd),'g.')
        source.x = xi;
        source.y = yi;
        set(handles.edit11_source_actual, 'string',
mat2str([xi+placement_error2(linInd) yi+placement_error1(linInd)]));
    end
end
disp('Select Destination');
[xi,yi,but] = ginput(1);
xi=round(xi);yi=round(yi);
if(xi<=handles.hsize && yi<=handles.vsize)
    if(GnetworkNodeType(yi,xi) == 1)
        linInd = sub2ind([handles.vsize handles.hsize], yi, xi);
        plot(xi+placement_error2(linInd),yi+placement_error1(linInd),'m.')
        dest.x = xi;
        dest.y = yi;
        set(handles.edit14_dest_actual, 'string',
mat2str([xi+placement_error2(linInd) yi+placement_error1(linInd)]));
    end
end
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Find Adjacency matrix required for gpsr_forward
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
range = handles.range;
sensors = find(GnetworkNodeType==1);
no_sensors = numel(sensors);
matrix_size = size(GnetworkNodeType);
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
no_anchors = numel(GanchorPos);

[row_cordn col_cordn] = find(GnetworkNodePos);
r =1;
if(isempty(adjmatrix1))
    for k = 1:no_nodes
        for i = floor(max(1,row_cordn(k)-
range)):ceil(min(matrix_size(1),(row_cordn(k)+range)))
            for j = floor(max(1,col_cordn(k)-
range)):ceil(min(matrix_size(2),(col_cordn(k)+range)))
                nodeIndex1 = sub2ind(size(GnetworkNodeType), i, j);
                nodeIndex2 = sub2ind(size(GnetworkNodeType), row_cordn(k),
col_cordn(k));
```

```
                if (GnetworkNodeType(i,j) ~= 0)
                    if (i ~= row_cordn(k) || j ~= col_cordn(k))
                        if(sqrt((abs(i+placement_error1(nodeIndex1)-
row_cordn(k)-placement_error1(nodeIndex2))^2 +
abs(j+placement_error2(nodeIndex1)-col_cordn(k)-
placement_error2(nodeIndex2))^2)) <= (handles.range))
                            Gpairs(r,1) =
GnetworkNodePos(row_cordn(k),col_cordn(k));
                            Gpairs(r,2) = GnetworkNodePos(i,j);
                            r = r + 1;
                        end
                    end
                end
            end
        end
    end
    adjmatrix1 = accumarray(Gpairs, 1);
    if(size(adjmatrix1,1) < no_nodes)
        adjmatrix1(no_nodes,1) = 0;
        adjmatrix1(1,no_nodes) = 0;
    end
end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

[not_present(:,2) not_present(:,1)] = find(GnetworkNodePos==0);




switch str{val}
    case 'GPSR-Using Location'
        p = struct('D',{},'L_p', {}, 'L_f', {}, 'e_0', {}, 'M',{});
        %          self = get(handles.edit11_source_actual,'String');
        %          source.x =
str2double(get(handles.edit11_source_actual,'String'));
        %          source.y =
str2double(get(handles.edit14_dest_actual,'String'));

        self.x = source.x;
        self.y = source.y;
        n_in.x = self.x;
        n_in.y = self.y;
        %          p(1).D = get(handles.edit12_vc_source,'String');
        %          p(1).D.x =
str2double(get(handles.edit12_vc_source,'String'));
        %          p(1).D.y =
str2double(get(handles.edit15_vc_dest,'String'));
```

```
        p(1).D.x = dest.x;
        p(1).D.y = dest.y;
        p(1).M = 'Greedy';
        global lp_flag;
        lp_flag = 0;
        while(flag==0)
            [p, flag, n_in]=gpsr_forward(p, n_in);
        end
        set(handles.edit13_route_hop,'String',route_hop);
        sourceInd = sub2ind(size(GnetworkNodeType), source.y, source.x);
        destInd = sub2ind(size(GnetworkNodeType), dest.y, dest.x);
        if (~(GnetworkNodePos(sourceInd) && GnetworkNodePos(destInd)))
            errordlg('Source or Destination not correct','Route Error');
            uiwait(gcf);
            return;
        end
        % %         n_in = node_Matrix(linearInd);
        %          n_in = self;
        %          p.M = 'Greedy';
        %          while(~flag)
        %              [p, n_in, flag] = gpsr_forward(p, n_in);
        %          end
        %          disp(['Hops taken ', num2str(route_hop)]);
    case 'Deterministic'
        VC_matrix = [];
        sourceInd = sub2ind(size(GnetworkNodeType), source.y, source.x);
        destInd = sub2ind(size(GnetworkNodeType), dest.y, dest.x);
        if(no_anchors>0)
            S_adj = sparse(adjmatrix1);
            for i = 1:no_anchors
                VC_matrix(i,:)=dijkstra_sp(S_adj,GanchorPos(i));
            end
            vcSource = VC_matrix(:,GnetworkNodePos(sourceInd));
            vcDest = VC_matrix(:,GnetworkNodePos(destInd));
            set(handles.edit12_vc_source, 'string', mat2str(vcSource));
            set(handles.edit15_vc_dest, 'string', mat2str(vcDest));
            h_AiAj = VC_matrix(2,GanchorPos(1));
            h_nAi = vcSource(1);
            h_nAj = vcSource(2);
            source.x = floor(h_AiAj/2)-floor((h_nAj-h_nAi)/2)+1;
            source.y = (h_nAi + h_nAj - h_AiAj)/2+1;
            h_nAi = vcDest(1);
            h_nAj = vcDest(2);
            dest.x = floor(h_AiAj/2)-floor((h_nAj-h_nAi)/2)+1;
            dest.y = (h_nAi + h_nAj - h_AiAj)/2+1;
        end
        route_hop = 0;
        if(get(handles.radiobutton6_rectangular_topology,'value')==1)
% Rectangular Network
            possible_directions(:,:,1) = [1 1 0; 2 0 1; -1 -1 0; -2 0 -1];
            possible_directions(:,:,2) = [1 1 0; -2 0 -1; -1 -1 0; 2 0 1];
            possible_directions(:,:,3) = [-1 -1 0; 2 0 1; 1 1 0; -2 0 -1];
            possible_directions(:,:,4) = [-1 -1 0; -2 0 1; 1 1 0; 2 0 1];
            possible_directions(:,:,5) = [2 0 1; 1 1 0;  -2 0 -1; -1 -1 0];
            possible_directions(:,:,6) = [2 0 1; -1 -1 0;  -2 0 -1; 1 1 0];
            possible_directions(:,:,7) = [-2 0 -1; 1 1 0;  2 0 1; -1 -1 0];
            possible_directions(:,:,8) = [-2 0 -1; -1 -1 0;  2 0 1; 1 1 0];
        elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 4;                      % Traingular Network
        elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 2;                         % hexagonal Network        p =
struct('D',{},'P_Route', {}, 'L_f', {}, 'e_0', {}, 'M',{});
        end


        self.x = source.x;
        self.y = source.y;
        n_in.x = self.x;
        n_in.y = self.y;
        p(1).D.x = dest.x;
        p(1).D.y = dest.y;


        if (~(GnetworkNodePos(sourceInd) && GnetworkNodePos(destInd)))
            errordlg('Source or Destination not correct','Route Error');
            uiwait(gcf);
            return;
        end
        set(handles.edit19_est_source, 'string', mat2str([source.x
source.y]));
        set(handles.edit20_est_dest, 'string', mat2str([dest.x dest.y]));
        horizontal_forward = p.D.x - source.x;
        vertical_forward = p.D.y - source.y;
        P(1,:) = [sign(horizontal_forward)*ones([1 abs(horizontal_forward)])
2*sign(vertical_forward)*ones([1 abs(vertical_forward)])];
        P(2,:) = [2*sign(vertical_forward)*ones([1 abs(vertical_forward)])
sign(horizontal_forward)*ones([1 abs(horizontal_forward)])];
        paths = random_paths(P);
        global GvoidVertices;
        global GvoidNo;
        %          [no_paths paths] = uperms(P, 5);
        no_change_required = Inf;
        min_route_hop =0;
        path_change = [];
        path_id = [];
        for ii = 1:size(paths,1)
            deter_directions = paths(ii,:)';
            path_addition = zeros(length(deter_directions),2);
            path_addition(deter_directions==1,:)=repmat([1
0],length(find(deter_directions==1)),1);
            path_addition(deter_directions==-1,:)=repmat([-1
0],length(find(deter_directions==-1)),1);
            path_addition(deter_directions==2,:)=repmat([0
1],length(find(deter_directions==2)),1);
            path_addition(deter_directions==-2,:)=repmat([0 -
1],length(find(deter_directions==-2)),1);
            path_addition = cumsum(path_addition,1);
            %             path_addition = [0 0;path_addition];
            a = repmat([source.x source.y],length(deter_directions),1);
            deter_route = a + path_addition;
            IN = [];
            %             for jj = 1: (GvoidNo -1)
            %                  x = 
inpolygon(deter_route(:,1),deter_route(:,2),GvoidVertices(:,1,jj),GvoidVertic
es(:,2,jj));
            %                  IN = cat(2,IN, x');
            %              end
            IN = ismember(deter_route,not_present,'rows');
            %              IN = 
inpolygon(deter_route(:,1),deter_route(:,2),GvoidVertices(:,1),GvoidVertices(
:,2));
            if(nnz(IN)<no_change_required)
                no_change_required = nnz(IN);
                path_id = ii;
                path_change = IN;
                selected_path = deter_route;
            end
            if(nnz(IN)>0)
                continue;
            else
                break;
            end
        end
        deter_directions = paths(path_id,:);
        st_IN = IN;
        st_path_change = path_change;
        st_deter_directions = deter_directions;
        st_selected_path = selected_path;
        if(nnz(path_change)~=0)
            min_route_hop = Inf;
            for kk =  1:size(possible_directions,3)
                IN = st_IN;
                path_change = st_path_change;
                selected_path = st_selected_path;
                deter_directions = st_deter_directions;
                ttl = 0;
                travel_flag = 0;
                while(nnz(IN)~=0)
                    temp = diff(path_change);
                    no_of_obstacles =  find(temp==1);
                    end_of_obstacles = find(temp==-1);
                    %               for ii = 1:length(no_of_obstacles)
                    if(travel_flag==0)
                        st_min_index_in = Inf;
                        st_min_index_end = Inf;
                        for ii = 1:length(no_of_obstacles)
                            n_in_dir_pos = no_of_obstacles(ii);%(1);
                            index_pos_n_in = 
mod(n_in_dir_pos,size(deter_directions,2));
                            index_pos_n_in(index_pos_n_in==0) = 
size(deter_directions,2);
                            end_of_curr_obstacle = end_of_obstacles(ii);%(1);
                            index_pos_end_curr = 
mod(end_of_curr_obstacle,size(deter_directions,2));
                            index_pos_end_curr(index_pos_end_curr==0) = 
size(deter_directions,2);
                            if(index_pos_n_in < st_min_index_in)
                                st_min_index_in = index_pos_n_in;
                                st_min_index_end = index_pos_end_curr;
                            end
                        end
                        index_pos_n_in = st_min_index_in;
                        index_pos_end_curr = st_min_index_end;
                    else
                        index_pos_n_in = index_pos_n_in+1;
                        index_pos_end_curr = index_pos_end_curr+2;
                    end
                    valid_directions = possible_directions(:,:,kk);
                    valid_directions(valid_directions(:,1) == -
1*deter_directions(index_pos_n_in),:)=[];
                    if(travel_flag ~= 1)
                        valid_directions(valid_directions(:,1) == -
1*deter_directions(index_pos_n_in+1),:)=[];
                    end
                    choose_direction =
repmat(selected_path(index_pos_n_in,:),size(valid_directions,1),1)+valid_dire
ctions(:,[2 3]);
                    IN = [];
                    %                       for jj = 1: (GvoidNo -1)
                    %                           x =
inpolygon(choose_direction(:,1),choose_direction(:,2),GvoidVertices(:,1,jj),G
voidVertices(:,2,jj));
                    %                           IN = cat(2,IN, x');
                    %                       end
                    %                       y = mod(find(IN==1),size(x,1));
                    %                       y(y==0) = size(x,1);
                    IN = ismember(choose_direction,not_present,'rows');
                    y = find(IN==1);
                    valid_directions(y,:)= [];
                    choose_direction(y,:) = [];
                    if(numel(find(choose_direction>handles.hsize))>=1 ||
numel(find(choose_direction>handles.vsize))>=1 ...
                            || numel(find(choose_direction<1))>=1)
                        temp_ind = [];
                        for zz = 1: size(choose_direction,1)
                            if((choose_direction(zz,1)>handles.hsize) ||
(choose_direction(zz,2)>handles.vsize)...
                                    || (choose_direction(zz,1)<1)||
(choose_direction(zz,2)<1))
                                temp_ind(zz) = zz;
                            end
                        end
                        temp_ind(temp_ind==0)=[];
                        choose_direction(temp_ind,:) = [];
                        valid_directions(temp_ind,:) = [];
                    end
                    if( size(valid_directions,1)==0) %index_pos_n_in ==
(index_pos_end_curr-2) &&
                        valid_directions=
possible_directions(possible_directions(:,1,kk) == -
1*deter_directions(index_pos_n_in+1),:,kk);
                        choose_direction =
repmat(selected_path(index_pos_n_in,:),size(valid_directions,1),1)+valid_dire
ctions(:,[2 3]);
                        travel_flag = 1;
                        st_valid_directions(1) = valid_directions(1,1);
                        st_valid_directions(2) = -
deter_directions(index_pos_n_in);
                    end
                    if(travel_flag ==1 && size(valid_directions,1)>1)% &&
index_pos_n_in~=1)

choose_direction(valid_directions(:,1)==st_valid_directions(2),:) = [];

valid_directions(valid_directions(:,1)==st_valid_directions(2),:)=[];
                    end
                    if(travel_flag==1 &&
~isempty(find(ismember(valid_directions(:,1),st_valid_directions)==0, 1)))
                        travel_flag = 0;

choose_direction(ismember(valid_directions(:,1),st_valid_directions),:) = [];

valid_directions(ismember(valid_directions(:,1),st_valid_directions),:)=[];
                    end
                    if(isempty(valid_directions))
                        valid_directions =
possible_directions(possible_directions(:,1,kk)==-
1*deter_directions(index_pos_n_in),:,kk);
                        choose_direction =
repmat(selected_path(index_pos_n_in,:),size(valid_directions,1),1)+valid_dire
ctions(:,[2 3]);
                    end
                    d = choose_direction - repmat([p.D.x p.D.y],
size(choose_direction,1),1);
                    d = sum(d.^2,2).^0.5;
                    d = find(d == min(d));

                    x = [deter_directions(1:index_pos_n_in)
valid_directions(d(1),1)
deter_directions(index_pos_n_in+1:index_pos_end_curr+1)...
                        -1*valid_directions(d(1),1)
deter_directions(index_pos_end_curr+2:end)];
                    deter_directions = x;
                    path_addition = zeros(length(deter_directions),2);
                    path_addition(deter_directions==1,:)=repmat([1
0],length(find(deter_directions==1)),1);
                    path_addition(deter_directions==-1,:)=repmat([-1
0],length(find(deter_directions==-1)),1);
                    path_addition(deter_directions==2,:)=repmat([0
1],length(find(deter_directions==2)),1);
                    path_addition(deter_directions==-2,:)=repmat([0 -
1],length(find(deter_directions==-2)),1);
                    path_addition = cumsum(path_addition,1);
                    %              path_addition = [0 0;path_addition];
                    a = repmat([source.x
source.y],length(deter_directions),1);
                    deter_route = a + path_addition;
                    selected_path = deter_route;

                    [u,I,J] = unique(deter_route, 'rows', 'first');
                    hasDuplicates = size(u,1) < size(deter_route,1);
                    while (hasDuplicates && travel_flag ==0)
```

```
                        %*******************************
                        ixDupRows = setdiff(1:size(deter_route,1), I);
                        dupRowValues = deter_route(ixDupRows,:);
                        if(hasDuplicates)
                            zz = size(dupRowValues,1);
                            dupIndex =
find(ismember(deter_route,dupRowValues(zz,:),'rows'));
                            inBetweenInd = dupIndex(1)+1:dupIndex(2);
                            deter_route(inBetweenInd)=[];
                            deter_directions(inBetweenInd) = [];
                        end
                        %**********************************
                        path_addition = zeros(length(deter_directions),2);
                        path_addition(deter_directions==1,:)=repmat([1
0],length(find(deter_directions==1)),1);
                        path_addition(deter_directions==-1,:)=repmat([-1
0],length(find(deter_directions==-1)),1);
                        path_addition(deter_directions==2,:)=repmat([0
1],length(find(deter_directions==2)),1);
                        path_addition(deter_directions==-2,:)=repmat([0 -
1],length(find(deter_directions==-2)),1);
                        path_addition = cumsum(path_addition,1);
                        %             path_addition = [0 0;path_addition];
                        a = repmat([source.x
source.y],length(deter_directions),1);
                        deter_route = a + path_addition;
                        [u,I,J] = unique(deter_route, 'rows', 'first');
                        hasDuplicates = size(u,1) < size(deter_route,1);
                    end


                    IN =[];
                    %                 for jj = 1: (GvoidNo -1)
                    %                     x =
inpolygon(deter_route(:,1),deter_route(:,2),GvoidVertices(:,1,jj),GvoidVertic
es(:,2,jj));
                    %                     IN = cat(2,IN, x');
                    %                 end
                    IN = ismember(deter_route,not_present,'rows');
                    path_change = IN;



                    %             end

%*****************************************************************
                    % For debugging each route

%*****************************************************************
                    hold off
                    updateTotalNodesAnchors( handles );
                    hold on
                    for ii = 1:(size(deter_route,1)-1)
                        line([deter_route(ii,1)
deter_route(ii+1,1)],[deter_route(ii,2) deter_route(ii+1,2)]);
                    end
                    hold off
```

```
%*****************************************************************
                    ttl = ttl+1;
                    if(nnz(IN)==0 || ttl>min_route_hop ||
ttl>(max(max(GnetworkNodePos)))/2 ) break; end
                end
                %                      [u,I,J] = unique(deter_route, 'rows',
'first');
                %                      hasDuplicates = size(u,1) <
size(deter_route,1);
                %                      while (hasDuplicates && travel_flag
==0)
                %                          %*******************************
                %                          ixDupRows =
setdiff(1:size(deter_route,1), I);
                %                          dupRowValues =
deter_route(ixDupRows,:);
                %                          if(hasDuplicates)
                %                              zz = size(dupRowValues,1);
                %                              dupIndex =
find(ismember(deter_route,dupRowValues(zz,:),'rows'));
                %                              inBetweenInd =
dupIndex(1)+1:dupIndex(2);
                %                              deter_route(inBetweenInd)=[];
                %                              deter_directions(inBetweenInd)
= [];
                %                          end
                %                          %**********************************
                %                          path_addition =
zeros(length(deter_directions),2);
                %
path_addition(find(deter_directions==1),:)=repmat([1
0],length(find(deter_directions==1)),1);
                %
path_addition(find(deter_directions==-1),:)=repmat([-1
0],length(find(deter_directions==-1)),1);
                %
path_addition(find(deter_directions==2),:)=repmat([0
1],length(find(deter_directions==2)),1);
                %
path_addition(find(deter_directions==-2),:)=repmat([0 -
1],length(find(deter_directions==-2)),1);
                %                          path_addition =
cumsum(path_addition,1);
                %                          a = repmat([source.x
source.y],length(deter_directions),1);
                %                          deter_route = a + path_addition;
                %                          [u,I,J] = unique(deter_route,
'rows', 'first');
                %                          hasDuplicates = size(u,1) <
size(deter_route,1);
                %                      end

                first_dest = find(ismember(selected_path, [p.D.x p.D.y],
'rows')==1);
                deter_directions([first_dest+1:end])=[];
                [lgt first last e] = SplitVec(deter_directions, [],
'length','first','last', 'firstelem');
                %        z = find(lgt==1);
                if (length(lgt)>2)
                    z = [];
                    for ii = 2:(length(lgt)-1)
                        if(lgt(ii)==1)
                            if(e(ii-1)+e(ii+1)==0);
                                s = min([lgt(ii-1) lgt(ii+1)]);
                                z = cat(2,z,[(first(ii)-s):(first(ii)-1)
(first(ii)+1):(first(ii)+s)]);
                            end
                        end
                    end
                    z = unique(z);
                    deter_directions(z) = [];
                end
                if(length(deter_directions)<min_route_hop && nnz(IN)==0)
                    min_route_hop = length(deter_directions);
                    route_hop = min_route_hop;
                    temp_deter_directions = deter_directions;
                end
            end
            if (min_route_hop ~= Inf)
                deter_directions = temp_deter_directions;
            end
        end

        deter_directions(deter_directions==0)=[];
        path_addition = zeros(length(deter_directions),2);
        path_addition(deter_directions==1,:)=repmat([1
0],length(find(deter_directions==1)),1);
        path_addition(deter_directions==-1,:)=repmat([-1
0],length(find(deter_directions==-1)),1);
        path_addition(deter_directions==2,:)=repmat([0
1],length(find(deter_directions==2)),1);
        path_addition(deter_directions==-2,:)=repmat([0 -
1],length(find(deter_directions==-2)),1);
        path_addition = cumsum(path_addition,1);
        path_addition = [0 0;path_addition];
        a = repmat([source.x source.y],length(deter_directions)+1,1);
        deter_route = a + path_addition;
        route_hop = length(deter_route)-1;
        hold off
        updateTotalNodesAnchors( handles );
        hold on;
        for ii = 1:(size(deter_route,1))
            nodeIndex1 = sub2ind(size(GnetworkNodeType), deter_route(ii,2),
deter_route(ii,1));
            nodeIndex2 = sub2ind(size(GnetworkNodeType), deter_route(ii+1,2),
deter_route(ii+1,1));
            line([deter_route(ii,1)+placement_error2(nodeIndex1)
deter_route(ii+1,1)+placement_error2(nodeIndex2)],[deter_route(ii,2)+placemen
t_error1(nodeIndex1) deter_route(ii+1,2)+placement_error1(nodeIndex2)]);
        end
        if (min_route_hop == Inf)
            disp('Packet Failure');
        end
        set(handles.edit13_route_hop,'String',route_hop);

        %        while(1)
        %             deterministic_forward();
        %        end
end


% --- Executes during object creation, after setting all properties.
function popupmenu4_routingmethod_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu4_routingmethod (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in pushbutton7_route.
function pushbutton7_route_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7_route (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit13_route_hop_Callback(hObject, eventdata, handles)
% hObject    handle to edit13_route_hop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13_route_hop as text
%        str2double(get(hObject,'String')) returns contents of
edit13_route_hop as a double



% --- Executes during object creation, after setting all properties.
function edit13_route_hop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13_route_hop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit14_dest_actual_Callback(hObject, eventdata, handles)
% hObject    handle to edit14_dest_actual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14_dest_actual as text
%        str2double(get(hObject,'String')) returns contents of
edit14_dest_actual as a double


% --- Executes during object creation, after setting all properties.
function edit14_dest_actual_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14_dest_actual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit15_vc_dest_Callback(hObject, eventdata, handles)
% hObject    handle to edit15_vc_dest (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15_vc_dest as text
%        str2double(get(hObject,'String')) returns contents of edit15_vc_dest
as a double


% --- Executes during object creation, after setting all properties.
function edit15_vc_dest_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15_vc_dest (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit16_placement_error_Callback(hObject, eventdata, handles)
% hObject    handle to edit16_placement_error (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit16_placement_error as
text
%        str2double(get(hObject,'String')) returns contents of
edit16_placement_error as a double
placement_error = str2double(get(handles.edit16_placement_error,'String'));
if(placement_error > 1)
    set(handles.edit16_placement_error,'String',0);
end

% --- Executes during object creation, after setting all properties.
function edit16_placement_error_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit16_placement_error (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit17_disabled_anchors_Callback(hObject, eventdata, handles)
% hObject    handle to edit17_disabled_anchors (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17_disabled_anchors as
text
%        str2double(get(hObject,'String')) returns contents of
edit17_disabled_anchors as a double


% --- Executes during object creation, after setting all properties.
function edit17_disabled_anchors_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit17_disabled_anchors (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit18_voidVertices_Callback(hObject, eventdata, handles)
% hObject    handle to edit18_voidVertices (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18_voidVertices as
text
%        str2double(get(hObject,'String')) returns contents of
edit18_voidVertices as a double


% --- Executes during object creation, after setting all properties.
function edit18_voidVertices_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18_voidVertices (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton8_set_placement_error.
function pushbutton8_set_placement_error_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton8_set_placement_error (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1= [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;

GanchorPos = [];
GvoidVertices = [];
GvoidNo = 1;

nodes = numel(GnetworkNodeType);
nodeRow = (1:nodes)';
nodeCol = (1:nodes)';
```

```
placement_error = str2double(get(handles.edit16_placement_error,'String'));
placement_error = 2*placement_error;
placement_error1 = -(placement_error/2) +
placement_error.*rand(size(nodeRow,1),1);
placement_error1 = round(placement_error1*100)/100;
placement_error2 = -(placement_error/2) +
placement_error.*rand(size(nodeCol,1),1);
placement_error2 = round(placement_error2*100)/100;

updateTotalNodesAnchors( handles );
set(handles.pushbutton1_undo,'visible','on');
% Update handles structure
guidata(hObject, handles);



% --- Executes on button press in pushbutton9_show_connectivity.
function pushbutton9_show_connectivity_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9_show_connectivity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error2;
global placement_error1;
%global GcorrectVC;

range = handles.range;
matrix_size = size(GnetworkNodeType);
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
[row_cordn col_cordn] = find(GnetworkNodePos);
r =1;
for k = 1:no_nodes
    for i = floor(max(1,row_cordn(k)-
range)):ceil(min(matrix_size(1),(row_cordn(k)+range)))
        for j = floor(max(1,col_cordn(k)-
range)):ceil(min(matrix_size(2),(col_cordn(k)+range)))
            nodeIndex1 = sub2ind(size(GnetworkNodeType), i, j);
            nodeIndex2 = sub2ind(size(GnetworkNodeType), row_cordn(k),
col_cordn(k));
            if (GnetworkNodeType(i,j) ~= 0)
                if (i ~= row_cordn(k) || j ~= col_cordn(k))
                    if(sqrt((abs(i+placement_error1(nodeIndex1)-row_cordn(k)-
placement_error1(nodeIndex2))^2 + abs(j+placement_error2(nodeIndex1)-
col_cordn(k)-placement_error2(nodeIndex2))^2)) <= (handles.range))
                        Gpairs(r,1) =
GnetworkNodePos(row_cordn(k),col_cordn(k));
                        Gpairs(r,2) = GnetworkNodePos(i,j);
                        r = r + 1;
                    end
                end
            end
        end
    end
end

no_anchors = numel(GanchorPos);
updateTotalNodesAnchors( handles );
global adjmatrix1;
adjmatrix1 = accumarray(Gpairs, 1);
if(size(adjmatrix1,1) < no_nodes)
    adjmatrix1(no_nodes,1) = 0;
    adjmatrix1(1,no_nodes) = 0;
end

if(get(handles.radiobutton5_none_topology,'value')~=1)

    GcorrectVC = zeros(size(GnetworkNodeType));

    temp = ones(handles.vsize-2,handles.hsize-2);
    temp = padfill(temp,0);
    border_elements = find(temp==0);
    for ii = 1:no_nodes
        node_pos = find(GnetworkNodePos==ii);
        neighbors = find(adjmatrix1(ii,:));
        if(nnz(ismember(border_elements,node_pos))>0)
            if(get(handles.radiobutton6_rectangular_topology,'value')==1)
max_neighbors = 3;                      % Rectangular Network
            elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 4;                      % Traingular Network
            elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 2;                        % hexagonal Network
            end
        else
            if(get(handles.radiobutton6_rectangular_topology,'value')==1)
max_neighbors = 4;                      % Rectangular Network
            elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 6;                      % Traingular Network
            elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 3;                        % hexagonal Network
            end
        end
        if(numel(neighbors) <= max_neighbors)
            GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
        else
            GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 0;
        end
    end
    for kk = 1:20
        for ii = 1:no_nodes
            node_pos = find(GnetworkNodePos==ii);
            neighbors = find(adjmatrix1(ii,:));
            neighbors_pos = ismember(GnetworkNodePos,neighbors);
            neighbors_pos = find(neighbors_pos);
            if(nnz(ismember(border_elements,node_pos))>0)
                if(get(handles.radiobutton6_rectangular_topology,'value')==1)
max_neighbors = 3;                      % Rectangular Network
```

```
elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 4;                       % Traingular Network

elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 2;                         % hexagonal Network
                end
            else
                if(get(handles.radiobutton6_rectangular_topology,'value')==1)
max_neighbors = 4;                       % Rectangular Network

elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 6;                       % Traingular Network

elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 3;                         % hexagonal Network
                end
            end
            if(GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))==1)
                continue;
            else
                if(numel(neighbors) <= max_neighbors)
                    GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                    continue;
                end
                redundant_neighbor =
GcorrectVC(ind2sub(size(GnetworkNodeType),neighbors_pos));
                %If no. of elements less than required links, not sure which
...
                % links to remove so continue;
                if(numel(find(redundant_neighbor==1))<max_neighbors ||
numel(find(redundant_neighbor==0))==0)
                    continue;
                else
                    redundant_neighbor =
neighbors(find(redundant_neighbor==0));
                    x = repmat(ii,1,length(redundant_neighbor));
                    redundant_neighbor_index = [redundant_neighbor;x]';
                    y = [];
                    for jj = 1:size(redundant_neighbor_index,1)
                        y((jj*2-
1:jj*2),:)=perms(redundant_neighbor_index(jj,:));
                    end
                    y = sub2ind(size(adjmatrix1),y(:,1),y(:,2));
                    adjmatrix1(y) = 0;
                    GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                end
            end
        end
        if (get(handles.radiobutton7_triangular_topology,'value')~=1)
            for ii = 1:no_nodes
                node_pos = find(GnetworkNodePos==ii);
                neighbors = find(adjmatrix1(ii,:));
                neighbors_pos = ismember(GnetworkNodePos,neighbors);
                neighbors_pos = find(neighbors_pos);
                if(nnz(ismember(border_elements,node_pos))>0)
if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 3;                        % Rectangular Network

elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 4;                        % Traingular Network

elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 2;                          % hexagonal Network
                    end
                else

if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 4;                        % Rectangular Network

elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 6;                        % Traingular Network

elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 3;                          % hexagonal Network
                    end
                end
                if(GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))==1)
                    continue;
                else
                    if(numel(neighbors) <= max_neighbors)
                        GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))
= 1;
                        continue;
                    end
                    redundant_neighbor =
GcorrectVC(ind2sub(size(GnetworkNodeType),neighbors_pos));
                    % if wrongly connected neighbor is neighbor of conrrectly
                    % connected neighbor remove link to wrongly connected
neighbor.
                    neighbors_one = neighbors;
                    neighbors_zero = neighbors;
                    neighbors_one(find(redundant_neighbor==0))=[];
                    neighbors_zero(find(redundant_neighbor==1))=[];
                    [neighbors_neighbor neighbor_index]=
find(adjmatrix1(neighbors_one',:)');
                    neighbor_index(neighbors_neighbor==ii) = [];
                    neighbors_neighbor(find(neighbors_neighbor==ii)) = [];
                    redundant_neighbor = [];

redundant_neighbor=neighbors_zero(find(ismember(neighbors_zero,neighbors_neig
hbor)==1));
                    if(numel(redundant_neighbor)==1 ||
(numel(redundant_neighbor)>=1 &&
get(handles.radiobutton7_triangular_topology,'value')~=1))
                        x = repmat(ii,1,length(redundant_neighbor));
                        redundant_neighbor_index = [redundant_neighbor;x]';
                        y = [];
                        for jj = 1:size(redundant_neighbor_index,1)
                            y((jj*2-
1:jj*2),:)=perms(redundant_neighbor_index(jj,:));
                        end
                        y = sub2ind(size(adjmatrix1),y(:,1),y(:,2));
                        adjmatrix1(y) = 0;
                        neighbors = find(adjmatrix1(ii,:));
                        if(numel(neighbors) <= max_neighbors)
                            
GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                        end
                    end
                end
            end
        end
        
        
        if (get(handles.radiobutton7_triangular_topology,'value')==1)
            %****************************************
            % Only for triangular network
            %***************************************
            for ii = 1:no_nodes
                node_pos = find(GnetworkNodePos==ii);
                neighbors = find(adjmatrix1(ii,:));
                neighbors_pos = ismember(GnetworkNodePos,neighbors);
                neighbors_pos = find(neighbors_pos);
                if(nnz(ismember(border_elements,node_pos))>0)
                    
if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 3;                    % Rectangular Network
                    
elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 4;                    % Traingular Network
                    
elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 2;                      % hexagonal Network
                    end
                else
                    
if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 4;                    % Rectangular Network
                    
elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 6;                    % Traingular Network
                    
elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 3;                      % hexagonal Network
                    end
                end
                redundant_neighbor = 
GcorrectVC(ind2sub(size(GnetworkNodeType),neighbors_pos));
                neighbors_one = neighbors;
                neighbors_zero = neighbors;
                neighbors_one(find(redundant_neighbor==0))=[];
                neighbors_zero(find(redundant_neighbor==1))=[];
                if(GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))==1 &&
numel(neighbors_zero)>=2)
                    
                    if(numel(neighbors_zero)==2)
```

```
                        [neighbors_one_neighbor neighbor_index]=
find(adjmatrix1(neighbors_zero(1),:)');
                        [neighbors_two_neighbor neighbor_index]=
find(adjmatrix1(neighbors_zero(2),:)');

neighbors_one_neighbor(neighbors_one_neighbor==ii)=[];

neighbors_two_neighbor(neighbors_one_neighbor==ii)=[];

neighbors_one_neighbor(find(ismember(neighbors_one_neighbor,neighbors_zero)))
=[];

neighbors_two_neighbor(find(ismember(neighbors_one_neighbor,neighbors_zero)))
=[];

neighbors(find(ismember(neighbors,neighbors_zero)))=[];

if(find(ismember(neighbors,intersect(neighbors_one_neighbor,neighbors_two_nei
ghbor))))

                            y = [];
                            y(1,:) = [neighbors_zero(1) neighbors_zero(2)];
                            y(2,:) = [neighbors_zero(2) neighbors_zero(1)];
                            if(~isempty(y))
                                y = sub2ind(size(adjmatrix1),y(:,1),y(:,2));
                                adjmatrix1(y) = 0;
                            end
                            node_pos =
find(GnetworkNodePos==neighbors_zero(1));
                            if(nnz(ismember(border_elements,node_pos))>0)
max_neighbors = 4;
                            else max_neighbors = 6;
                            end
                            if(numel(find(adjmatrix1(neighbors_zero(1),:)))<=
max_neighbors)

GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                            end
                            node_pos =
find(GnetworkNodePos==neighbors_zero(2));
                            if(nnz(ismember(border_elements,node_pos))>0)
max_neighbors = 4;
                            else max_neighbors = 6;
                            end
                            if
(numel(find(adjmatrix1(neighbors_zero(2),:)))<= max_neighbors)

GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                            end
                        end
                    elseif(numel(neighbors_zero)==3)
                        [neighbors_one_neighbor neighbor_index]=
find(adjmatrix1(neighbors_zero(1),:)');
                        [neighbors_two_neighbor neighbor_index]=
find(adjmatrix1(neighbors_zero(2),:)');
                        [neighbors_three_neighbor neighbor_index]=
find(adjmatrix1(neighbors_zero(3),:)');
```

```
neighbors_one_neighbor(neighbors_one_neighbor==ii)=[];

neighbors_two_neighbor(neighbors_one_neighbor==ii)=[];

neighbors_three_neighbor(neighbors_three_neighbor==ii)=[];

neighbors_one_neighbor(find(ismember(neighbors_one_neighbor,neighbors_zero)))
=[];

neighbors_two_neighbor(find(ismember(neighbors_one_neighbor,neighbors_zero)))
=[];

neighbors_three_neighbor(find(ismember(neighbors_three_neighbor,neighbors_zer
o)))=[];
                        one_two =
intersect(neighbors_one_neighbor,neighbors_two_neighbor);
                        two_three =
intersect(neighbors_three_neighbor,neighbors_two_neighbor);
                        three_one =
intersect(neighbors_three_neighbor,neighbors_one_neighbor);
                        y = [];
                        if(isempty(one_two))
                            y(1,:) = [neighbors_zero(1) neighbors_zero(2)];
                            y(2,:) = [neighbors_zero(2) neighbors_zero(1)];
                        elseif(isempty(two_three))
                            y(1,:) = [neighbors_zero(2) neighbors_zero(3)];
                            y(2,:) = [neighbors_zero(3) neighbors_zero(2)];
                        elseif(isempty(three_one))
                            y(1,:) = [neighbors_zero(1) neighbors_zero(3)];
                            y(2,:) = [neighbors_zero(3) neighbors_zero(1)];
                        end
                        if(~isempty(y))
                            y = sub2ind(size(adjmatrix1),y(:,1),y(:,2));
                            adjmatrix1(y) = 0;
                        end
                        node_pos = find(GnetworkNodePos==neighbors_zero(1));
                        if(nnz(ismember(border_elements,node_pos))>0)
max_neighbors = 4;
                        else max_neighbors = 6;
                        end
                        if(numel(find(adjmatrix1(neighbors_zero(1),:)))<=
max_neighbors)

GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                        end
                        node_pos = find(GnetworkNodePos==neighbors_zero(2));
                        if(nnz(ismember(border_elements,node_pos))>0)
max_neighbors = 4;
                        else max_neighbors = 6;
                        end
                        if (numel(find(adjmatrix1(neighbors_zero(2),:)))<=
max_neighbors)

GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                        end
                    end
```

```
                end
            end


        elseif (get(handles.radiobutton6_rectangular_topology,'value')==1  )
            %****************************************
            % Only for rectangular network
            %***************************************
            for ii = 1:no_nodes
                node_pos = find(GnetworkNodePos==ii);
                neighbors = find(adjmatrix1(ii,:));
                neighbors_pos = ismember(GnetworkNodePos,neighbors);
                neighbors_pos = find(neighbors_pos);
                if(nnz(ismember(border_elements,node_pos))>0)

if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 3;                    % Rectangular Network

elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 4;                    % Traingular Network

elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 2;                      % hexagonal Network
                    end
                else

if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 4;                    % Rectangular Network

elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 6;                    % Traingular Network

elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 3;                      % hexagonal Network
                    end
                end
                if(GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))==1)
                    continue;
                else
                    if(numel(neighbors) <= max_neighbors)
                        GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))
= 1;
                        continue;
                    end
                    redundant_neighbor =
GcorrectVC(ind2sub(size(GnetworkNodeType),neighbors_pos));
                    neighbors_one = neighbors;
                    neighbors_zero = neighbors;
                    neighbors_one(find(redundant_neighbor==0))=[];
                    if ( length(neighbors_one) >= (max_neighbors-2))%  &&
length(neighbors_pos)<=(max_neighbors+2))
                        neighbors_zero(find(redundant_neighbor==1))=[];
                        [neighbors_neighbor neighbor_index]=
find(adjmatrix1(neighbors_zero',:)');
                        neighbor_index(neighbors_neighbor==ii) = [];
                        neighbors_neighbor(find(neighbors_neighbor==ii)) =
[];
                        [neighbors_neighbor_neighbor
neighbor_neighbor_index]= find(adjmatrix1(neighbors_neighbor,:)');
                        redundant_neighbor = [];
                        correct_connected = neighbors_one;
                        for ll = 1:length(neighbors_one)
                            temp =
find(ismember(neighbors_neighbor_neighbor,neighbors_one(1,ll)));
                            temp =
neighbors_neighbor(neighbor_neighbor_index(temp));
                            temp =
neighbor_index(find(ismember(neighbors_neighbor,temp)));
                            temp = neighbors_zero(temp');
                            if(~isempty(temp))

correct_connected=cat(2,correct_connected,temp);
                            end
                        end
                        correct_connected = unique(correct_connected);
                        if(numel(correct_connected) == max_neighbors)

redundant_neighbor=neighbors_zero(find(ismember(neighbors_zero,correct_connec
ted)==0));
                            x = repmat(ii,1,length(redundant_neighbor));
                            redundant_neighbor_index =
[redundant_neighbor;x]';
                            y = [];
                            for jj = 1:size(redundant_neighbor_index,1)
                                y((jj*2-
1:jj*2),:)=perms(redundant_neighbor_index(jj,:));
                            end
                            y = sub2ind(size(adjmatrix1),y(:,1),y(:,2));
                            adjmatrix1(y) = 0;

GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                        end
                    end
                end
            end

        else
            %****************************************
            % Only for hexagonal network
            %***************************************
            for ii = 1:no_nodes
                node_pos = find(GnetworkNodePos==ii);
                neighbors = find(adjmatrix1(ii,:));
                neighbors_pos = ismember(GnetworkNodePos,neighbors);
                neighbors_pos = find(neighbors_pos);
                if(nnz(ismember(border_elements,node_pos))>0)

if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 3;                    % Rectangular Network

elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 4;                          % Traingular Network
```

```
elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 2;                    % hexagonal Network
                    end
                else

if(get(handles.radiobutton6_rectangular_topology,'value')==1)   max_neighbors
= 4;                    % Rectangular Network

elseif(get(handles.radiobutton7_triangular_topology,'value')==1)
max_neighbors = 6;                    % Traingular Network

elseif(get(handles.radiobutton8_hexagonal_topology,'value')==1)
max_neighbors = 3;                    % hexagonal Network
                    end
                end
                if(GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))==1)
                    continue;
                else
                    if(numel(neighbors) <= max_neighbors)
                        GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos))
= 1;

                        continue;
                    end
                    redundant_neighbor =
GcorrectVC(ind2sub(size(GnetworkNodeType),neighbors_pos));
                    neighbors_one = neighbors;
                    neighbors_zero = neighbors;
                    neighbors_one(find(redundant_neighbor==0))=[];
                    neighbors_zero(find(redundant_neighbor==1))=[];
                    [neighbors_neighbor neighbor_index]=
find(adjmatrix1(neighbors_one',:)');
                    neighbor_index(find(neighbors_neighbor==ii)) = [];
                    neighbors_neighbor(find(neighbors_neighbor==ii)) = [];
                    neighbors_neighbor = sort(neighbors_neighbor);
                    neighbors_neighbors_pos =
ismember(GnetworkNodePos,neighbors_neighbor);
                    neighbors_neighbors_pos = find(neighbors_neighbors_pos);
                    redundant_neighbor_neighbor =
GcorrectVC(ind2sub(size(GnetworkNodeType),neighbors_neighbors_pos));
                    neighbors_neighbors_one = neighbors_neighbor;

neighbors_neighbors_one(find(redundant_neighbor_neighbor==0))=[];
                    [neighbors_neighbor_neighbor neighbor_neighbor_index]=
find(adjmatrix1(neighbors_neighbors_one,:)');
                    redundant_neighbor = [];
                    neighbors_neighbor_neighbor =
unique(neighbors_neighbor_neighbor);

redundant_neighbor=neighbors_zero(find(ismember(neighbors_zero,neighbors_neig
hbor_neighbor)==1));
                    if(~isempty(redundant_neighbor))
                        x = repmat(ii,1,length(redundant_neighbor));
                        redundant_neighbor_index = [redundant_neighbor;x]';
                        y = [];
                        for jj = 1:size(redundant_neighbor_index,1)
```

```
                            y((jj*2-
1:jj*2),:)=perms(redundant_neighbor_index(jj,:));
                        end
                        y = sub2ind(size(adjmatrix1),y(:,1),y(:,2));
                        adjmatrix1(y) = 0;
                        neighbors = find(adjmatrix1(ii,:));
                        if(numel(neighbors) <= max_neighbors)

GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                        end
                    end
                    [neighbors_neighbor neighbor_index]=
find(adjmatrix1(neighbors_zero',:)');
                    neighbor_index(find(neighbors_neighbor==ii)) = [];
                    neighbors_neighbor(neighbors_neighbor==ii) = [];
                    neighbors_neighbor_neighbor =
sort(neighbors_neighbor_neighbor);
                    neighbors_neighbors_pos =
ismember(GnetworkNodePos,neighbors_neighbor_neighbor);
                    neighbors_neighbors_pos = find(neighbors_neighbors_pos);
                    redundant_neighbor_neighbor =
GcorrectVC(ind2sub(size(GnetworkNodeType),neighbors_neighbors_pos));
                    neighbors_neighbors_one = neighbors_neighbor_neighbor;

neighbors_neighbor_neighbor(find(redundant_neighbor_neighbor==0))=[];
                    redundant_neighbor = [];

redundant_neighbor=neighbor_index(find(ismember(neighbors_neighbor,neighbors_
neighbor_neighbor)==1));
                    redundant_neighbor=neighbors_zero(redundant_neighbor);
                    redundant_neighbor = unique(redundant_neighbor);
                    if(~isempty(redundant_neighbor))
                        x = repmat(ii,1,length(redundant_neighbor));
                        redundant_neighbor_index = [redundant_neighbor;x]';
                        y = [];
                        for jj = 1:size(redundant_neighbor_index,1)
                            y((jj*2-
1:jj*2),:)=perms(redundant_neighbor_index(jj,:));
                        end
                        y = sub2ind(size(adjmatrix1),y(:,1),y(:,2));
                        adjmatrix1(y) = 0;
                        neighbors = find(adjmatrix1(ii,:));
                        if(numel(neighbors) <= max_neighbors)

GcorrectVC(ind2sub(size(GnetworkNodeType),node_pos)) = 1;
                        end
                    end
                end
            end
        end
    end
end

Gpairs = [];
[Gpairs(:,1) Gpairs(:,2)] = find(adjmatrix1);
```

```
hold on;
u_pairs = unique(Gpairs,'rows');

for i = 1:length(Gpairs)
    nodeIndex1 = sub2ind(size(GnetworkNodeType), row_cordn(u_pairs(i,1)),
col_cordn(u_pairs(i,1)));
    nodeIndex2 = sub2ind(size(GnetworkNodeType), row_cordn(u_pairs(i,2)),
col_cordn(u_pairs(i,2)));
    plot([col_cordn(u_pairs(i,1))+placement_error2(nodeIndex1)
col_cordn(u_pairs(i,2))+placement_error2(nodeIndex2)],[row_cordn(u_pairs(i,1)
)+placement_error1(nodeIndex1)...
        row_cordn(u_pairs(i,2))+placement_error1(nodeIndex2)]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% To emphasize nodes on top of lines
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[nodeRow nodeCol] = find(handles.Topology.nodeType==1);
oneInd = find(handles.Topology.nodeType==1);
nodeCol = nodeCol + placement_error2(oneInd);
nodeRow = nodeRow + placement_error1(oneInd);
plot(nodeCol,nodeRow,'k.');
set(handles.edit8_numofnodes,'String',size(nodeCol,1));

[nodeRow nodeCol] = find(handles.Topology.nodeType==2);
twoInd = find(handles.Topology.nodeType==2);
nodeCol = nodeCol + placement_error2(twoInd);
nodeRow = nodeRow + placement_error1(twoInd);
plot(nodeCol,nodeRow,'r.');
set(handles.edit10_numofanchors,'String',size(nodeCol,1));
axis([0.5 handles.hsize+1 0.5 handles.vsize+1]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hold off
% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in checkbox1_svd.
function checkbox1_svd_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1_svd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1_svd


% --- Executes on button press in checkbox2_mds.
function checkbox2_mds_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox2_mds (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox2_mds
% --- Executes on button press in checkbox3_dvc.
function checkbox3_dvc_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox3_dvc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox3_dvc




% --- Executes when selected object is changed in uipanel11_topology_control.
function uipanel11_topology_control_SelectionChangeFcn(hObject, eventdata,
handles)
% hObject    handle to the selected object in uipanel11_topology_control
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was
selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)

% Update handles structure
guidata(hObject, handles);




function edit19_est_source_Callback(hObject, eventdata, handles)
% hObject    handle to edit19_est_source (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19_est_source as text
%        str2double(get(hObject,'String')) returns contents of
edit19_est_source as a double


% --- Executes during object creation, after setting all properties.
function edit19_est_source_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19_est_source (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit20_est_dest_Callback(hObject, eventdata, handles)
% hObject    handle to edit20_est_dest (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit20_est_dest as text
%        str2double(get(hObject,'String')) returns contents of
edit20_est_dest as a double


% --- Executes during object creation, after setting all properties.
function edit20_est_dest_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20_est_dest (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in togglebutton1_placeNode.
function togglebutton1_placeNode_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1_placeNode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton1_placeNode
global GanchorPos;
global GnetworkNodeType;
% Global variable for network - easy for debugging
global GnetworkNodePos;
global placement_error1;
global placement_error2;
global GvoidVertices;
global GvoidNo;
global adjmatrix1;
adjmatrix1= [];

handles.anchorPosBackup = GanchorPos;
handles.topologyBackup = handles.Topology;
handles.placement_error1_backup = placement_error1;
handles.placement_error2_backup = placement_error2;
handles.voidVerticesBackup = GvoidVertices;
handles.voidNoBbackup = GvoidNo;
handles.rangeBackup = handles.range;



but = 1;
hold on;
xi = handles.hsize + 1.1234;                              % To avoid the
error "Undefined variable xi"
yi = handles.vsize + 1.1234;                              % To avoid the
error "Undefined variable yi"
while but ==1
```

```
    xi = xi;
    yi = yi;
    button_state = get(hObject, 'Value');
    if button_state == get(hObject,'Max')
        [xi,yi] = ginput(1);
        xi=round(xi);yi=round(yi);
        linInd = sub2ind([handles.vsize handles.hsize], yi, xi);
        if(xi<=handles.hsize && yi<=handles.vsize)

plot(xi+placement_error2(linInd),yi+placement_error1(linInd),'b.')
            drawnow;
            temp = GnetworkNodeType(yi,xi);
            GnetworkNodeType(yi,xi)=1;
        end
        %       end
    elseif button_state == get(hObject,'Min')
        if(xi<=handles.hsize && yi<=handles.vsize)
            plot(xi,yi,'w.')
            GnetworkNodeType(yi,xi)=temp;
        end
        break;
    end
end
hold off;
anchor_ind = [];
for ii = 1:numel(GanchorPos);
    anchor_ind(ii) = find(GnetworkNodePos==GanchorPos(ii));
end
nodes = find(GnetworkNodeType);
no_nodes = numel(nodes);
node_ind = 1:no_nodes;
GnetworkNodePos = zeros(size(GnetworkNodeType));
GnetworkNodePos(nodes)= node_ind;                                            %
Define Node positions
GanchorPos = GnetworkNodePos(anchor_ind);
GanchorPos = unique_no_sort(GanchorPos);

handles.Topology.nodeType = GnetworkNodeType;
handles.Topology.nodePos = GnetworkNodePos;
updateTotalNodesAnchors( handles );                         % Function
updates the Total Nodes and Total ANchors box in GUI
% Update handles structure
set(handles.pushbutton1_undo,'visible','on');
guidata(hObject, handles);
```