Simulation of Position-Based Routing Algorithms in Wireless Ad hoc Networks with Irregular Transmission Ranges

Hai Xiao Chai

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March 2003

0-612-77706-5

Abstract

Simulation of Position-Based Routing Algorithms in Wireless Ad hoc Networks with Irregular Transmission Ranges

Hai Xiao Chai

In wireless mobile ad hoc networks (MANETs), there are two main categories of routing protocols: flooding-based and position-based. Flooding-based protocols waste precious bandwidth in wireless networks; while position-based protocols attempt to reduce the bandwidth used for control traffic. Many of the position-based protocols use a unit disk graph model for the network and thus implicitly assume a uniform transmission range for mobile hosts. However, this assumption may not be true in reality. Due to the irregular transmission range, it may not be as straightforward to extract a planar and connected subgraph on which to perform routing. In [1], a position-based routing algorithm that can handle irregular transmission range and guarantee delivery of messages is presented. In this project, we simulate the irregular transmission range scenario and compare the performance of five routing protocols in this scenario: Dijkstra's shortest path[4], Greedy routing[13], Perimeter routing[3], GPSR[10] and RPBR[1].

Acknowledgements

The simulations in this project are totally based on the theory in the paper "Robust Position-Based Routing in Wireless Ad-hoc Networks with Irregular Transmission Ranges" by Lali Barriere, Pierre Fraigniaud, Lata Narayanan and Jaroslav Opatrny. In that paper, the proof of the correctness of the algorithm is provided. I really appreciate the instruction and help from Dr. Lata Narayanan. Without her support, I would not have been able to correctly implement the routing algorithms simulated in this project.

Table of Contents

List of Figures

1. Introduction

An *ad hoc* network is a kind of a local area network (LAN). In Latin, ad hoc literally means "for this", meaning "for this purpose only", and thus implies temporary. An ad hoc network can be temporarily formed by mobile or portable devices when there are messages that need to be delivered. Devices use the network to communicate with each other by wireless links. Two nodes that are within each other's transmission range are connected neighbors in the network. Not all nodes are within each other's transmission ranges; this implies the need for a multi-hop routing protocol, where intermediate nodes must forward packets on behalf of the other nodes. An ad hoc network does not have a fixed topology. As the mobile devices move, the topology can change.

Routing in wireless mobile ad hoc networks is different from routing in wired networks. Wired networks usually have a fixed topology (such as bus, ring and star topology) and have adequate bandwidth. They don't need frequent updates of routing tables, and in any case, broadcasting control packets uses only a small amount of the total bandwidth. In contrast, ad hoc wireless networks have a changing topology, at the same time as having limited bandwidth. Ad hoc networks are also different from other infrastructure wireless network systems such as WLANS (wireless LANs); an ad hoc wireless network is an infrastructureless network. Each mobile node operates not only as a host but also as a router, forwarding packets for other mobile nodes in the network that may not be within direct wireless transmission range of each other. Each node participates in an ad hoc routing protocol that allows it to discover multi-hop paths through the network to any other node.

There are mainly two types of routing protocols in ad hoc wireless networks: flooding-based and position-based. Flooding-based routing protocols broadcast routing discovery messages over the entire network in order to find a route from source to destination. Each node may cache a routing table. A stale routing table can cause routing failures, and so, routing tables have to be up-to-date. Therefore, control messages are broadcasted periodically or whenever a change of topology is detected. In a wireless mobile network, bandwidth is limited and the topology keeps changing, therefore the cost to update the routing tables of each node can be prohibitively high.

To avoid wasting bandwidth, one approach is to use position-based routing protocols. In these protocols, it is assumed that each node knows its own coordinates and the destination node's coordinates. With this additional information, all the nodes locally calculate the next hop to the destination node, or limit the extent of flooding. Many sophisticated position-based routing protocols (such as perimeter routing[3] and GPSR[10]) use a unit disk graph model of the network. They extract a planar subgraph from the original graph and then find a route in it. However, the unit disk graph model is valid only if all nodes have *regular* transmission ranges, and moreover, they have the *same* transmission range. Regular transmission range means that the area reachable by a wireless transmitter is a circle of radius *r*, centered at the transmitter (see Figure1.1). However in reality, the transmission range may be *irregular* as shown in Figure 1.2. Additionally, there may be a slight variation in the power employed by different transmitters, so that the transmission ranges of different transmitters may not be exactly identical.

Most of the position-based routing protocols mentioned earlier fail if the transmission ranges are irregular or not identical. In particular, the extracted subgraph may not be planar or it may be disconnected. The RPBR[1] algorithm simulated in this project can guarantee the extraction of a planar and connected subgraph and thus guarantee delivery of packets. In the simulation, we study the performance of the RPBR algorithm and compare it with other well-known routing algorithms. In the results of the simulation, we will see that the performance of RPBR algorithm is as good as GPSR, and it never fails in an irregular transmission environment.

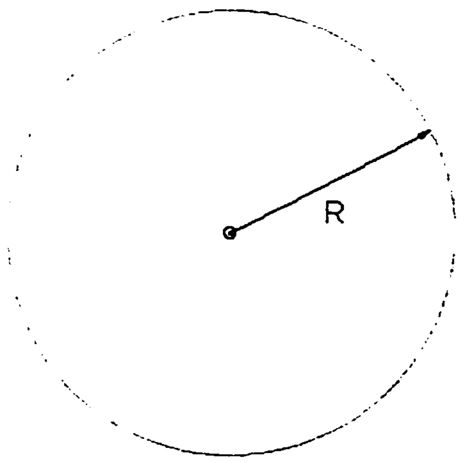


Fig 1.1 The transmission range is a disk with radius *R*.

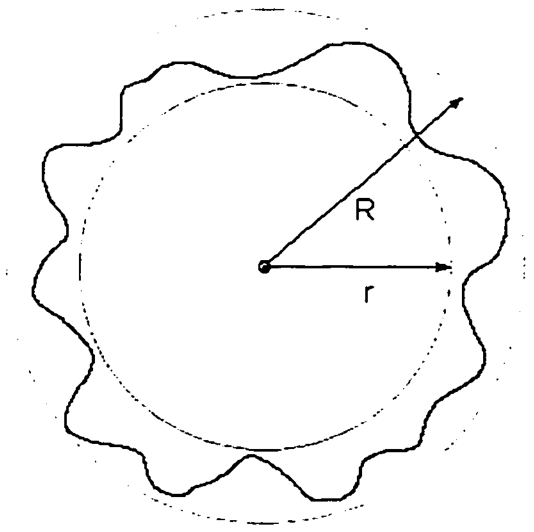


Fig 1.2 The transmission range varies between *r* and *R*.

In the next two sections, we describe the main flooding-based protocols as well as position-based protocols proposed in the literature.

1.1 Flooding-based Routing Protocols

In this section, we will discuss three flooding-based routing protocols in ad hoc wireless networks. They are Dynamic Source Routing (DSR) [9], Destination-Sequenced Distance-Vector Routing (DSDV) [17] and Ad Hoc On-Demand Distance Vector (AODV) [16]. Of there, DSR and AODV are currently candidates for acceptance as standards for routing in ad hoc networks.

1.1.1 Dynamic Source Routing (DSR)

DSR[9] uses source routing rather than hop-by-hop routing, with each packet to be routed carrying in its header the complete, ordered list of nodes through which the packet should pass. The key advantage of source routing is that intermediate nodes do not need to maintain up-to-date routing information in order to route the packets they forward, since the packets themselves already contain the entire routing path. This protocol eliminates the need for broadcasting a periodic route advertisement and neighbor detection packets.

The DSR protocol consists of two mechanisms: Route Discovery and Route Maintenance. Route Discovery is the mechanism by which a source node *S* wishing to send a packet to a destination *D* obtains a source route to *D*. To perform a Route Discovery, the source node *S* broadcasts a ROUTE REQUEST packet that is flooded through the network in a controlled manner and is answered by a ROUTE REPLY packet from either the destination node or another node that knows a route to the destination. To reduce the cost of Route Discovery, each node maintains a cache of source routes it has learned or overheard, which are aggressively used to limit the frequency of propagation of ROUTE

REQUESTs. Route Maintenance is the mechanism by which a packet's sender *S* detects if the network topology has changed such that it can no longer use its route to the destination *D* because the nodes listed in the route have moved out of range of each other. When Route Maintenance detects a source route is broken, *S* is notified with a ROUTE ERROR packet. The sender *S* can then attempt to use any other route to *D* already in its cache or invoke Route Discovery again to find a new route.

1.1.2 Destination-Sequenced Distance-Vector Routing (DSDV)

DSDV[17] is very similar to the Routing Information Protocol (RIP) [7]. In DSDV, each node keeps a routing table, and packets are routed in the ad hoc network using these routing tables. A routing table is actually a list of the addresses of all nodes in the network. For each address, the table indicates the next hop node.

Since in ad-hoc wireless networks the hosts keep moving, the routing tables need to be kept up-to-date. Whenever the network topology changes, as well as periodically, each node broadcasts a routing table update packet to other nodes. For some reasons such as packets getting delayed or lost, old update packets may be received later than more recent one. To avoid stale update packets messing up the routing table, each update packet has a sequence number given by the original node. When a node receives an update packet, it checks whether the sequence number is equal to or greater than the sequence number already in the routing table. Updates are accepted only if the new sequence number is greater than the old one, or if the sequence number is the same, but the hop metric is better. Otherwise the update packet is ignored.

1.1.3 Ad Hoc On-Demand Distance Vector (AODV)

AODV[16] is essentially a combination of both DSR[9] and DSDV[17]. It borrows the basic on-demand mechanism of Route Discovery and Route Maintenance from DSR, plus the use of hop-by-hop routing, sequence numbers, and periodic beacons from DSDV. When the source node *S* needs a route to some destination node *D*, it broadcasts a ROUTE REQUEST message to its neighbors, including the last known sequence number for that destination. The ROUTE REQUEST is flooded through the network in a controlled manner until it reaches a node that knows a route to the destination. Each node that forwards the ROUTE REQUEST creates a reverse route going back to node *S*. When the ROUTE REQUEST reaches a node with a route to *D*, that node generates a ROUTE REPLY that contains the number of hops necessary to reach *D* and the most recent sequence number for *D*. Each node that participates in forwarding this REPLY back to node *S* (the originator of the ROUTE REQUEST) creates a forward route to *D*. The state created in each node along the path from *S* to *D* is hop-by-hop state; that is, each node remembers only the next hop but not the entire route, as in source routing. In order to maintain the route, AODV normally requires that each node periodically send a HELLO message. Failure to receive three consecutive HELLO messages from a neighbor is taken as an indication that the link to the neighbor is broken. AODV also suggests that a node may use physical layer or link layer methods to detect link breakages. When a link goes down, any upstream node that has recently forwarded packets to a destination using that link is notified via an UNSOLICITED ROUTE REPLY containing an infinite metric for that destination. When a node receives such a ROUTE REPLY, it must discover a new route to the destination using Route Discovery again.

1.2 Position-based routing protocols

In the previous section, we discussed flooding-based routing protocols. They flood scarce bandwidth in ad-hoc wireless networks with control packets. This is obviously a waste of the resource. Position-based routing protocols don't have such a flaw. In this section, we discuss some position-based routing protocols. This type of protocol assumes that each node knows its own location, the locations of its direct neighbors, and that of the destination. However, nodes do not need to know the topology of the entire network, or locations of any other nodes. The next hop in the route is calculated locally in a distributed manner.

Most position-based protocols use a unit disk graph to model the ad hoc network and extract a planar and connected subgraph from that. The following notation will be used in this report in order to explain these protocols.

<u>Notation:</u>

$G = (V, E)$

  G – a geometric undirected graph (A geometric graph is a graph drawn in the plane such that its vertices are points in general position and its edges are straight-line segments)

  V – set of vertices

  E – set of edges (direct communication links)

$\{u, v\}$ – the edge between u and v

$d(u, v)$ – Euclidean distance between u and v

$B(u, \rho)$ – a ball with center u and radius ρ

$D_{u,v}$ – the disk determined by u and v with $d(u, v)$ as diameter

A geometric graph is called a unit disk graph if for any two vertices $u,v \in V$, the edge $\{u, v\} \in E$ if $d(u,v) \leq 1$. Given a geometric graph G, its *Gabriel graph GG(G)* is a subgraph in which the edge $\{u,v\}$ is removed if $D_{u,v}$ contains any other nodes [5]. See Figure 1.3 for an illustration. Since the Gabriel graph of a connected unit disk graph is guaranteed to be planar and connected [3], it is used widely in position-based protocols.

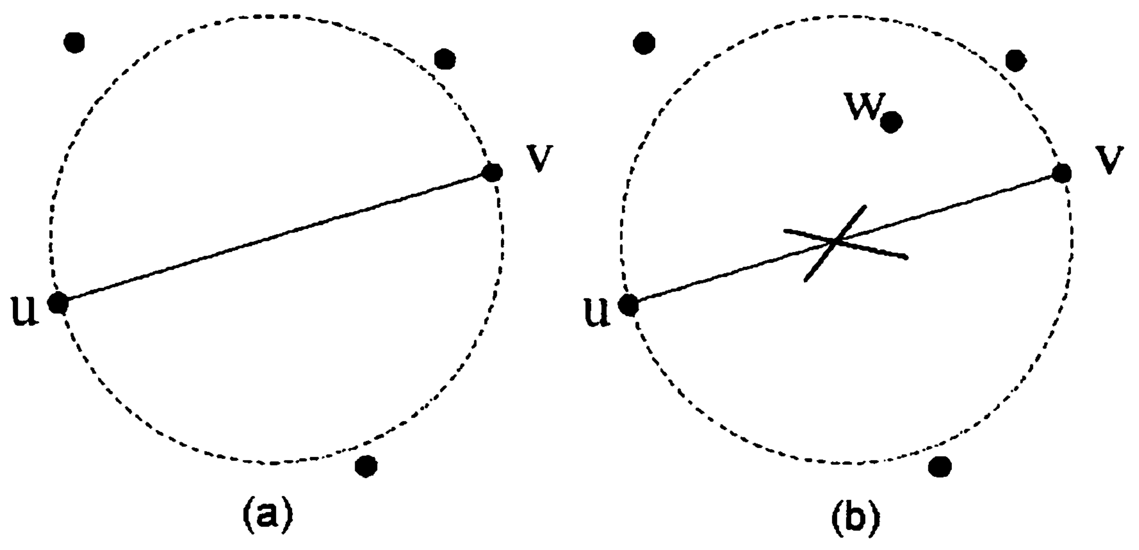


Fig 1.3 In (a), $\{u, v\}$ is retained in the Gabriel graph because there is no other node in $D_{u,v}$; in (b), $\{u, v\}$ is not retained in the Gabriel graph because there is a node w in $D_{u,v}$.

1.2.1 Location-Aided Routing (LAR)

LAR [12] is a modification of the DSR protocol. It still floods route discovery control packets over the network but limits the flooding region with the help of GPS. Hence the overhead of control packets is reduced. LAR works on demand. When the source node S wants to send a message to a destination node D, with the help of GPS, it defines a rectangular region (Request Zone) within which the flooding is limited. It also considers that the destination node is moving. The source node stores the speed of the destination node as well, and estimates where it will possibly be later (Expected Zone). It defines the Request Zone accordingly. Fig 1.4 shows an example.

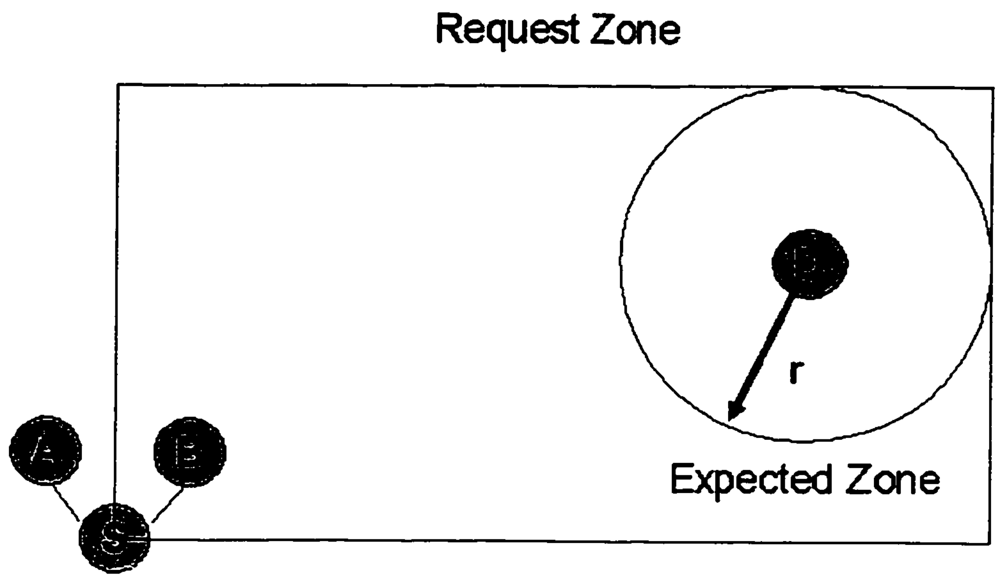


Fig 1.4 *S* will send control messages to *B* but not to *A* because *A* is not in the Request Zone. The Request Zone includes the Expected Zone.

1.2.2 Greedy Routing

Greedy routing [13] is very straightforward. Each node in ad-hoc networks knows its direct neighbor's coordinate. The source node *S* originates a packet that includes the coordinates of the destination node *D*. During forwarding, each node selects the closest node to *D* in its neighbor list until the packet reaches the destination node *D*. This algorithm is very efficient but it may fail while a node itself is the closest node to the destination node *D* in its neighbor list. Fig 1.5 gives two examples, one where greedy routing is successful and one where it fails.

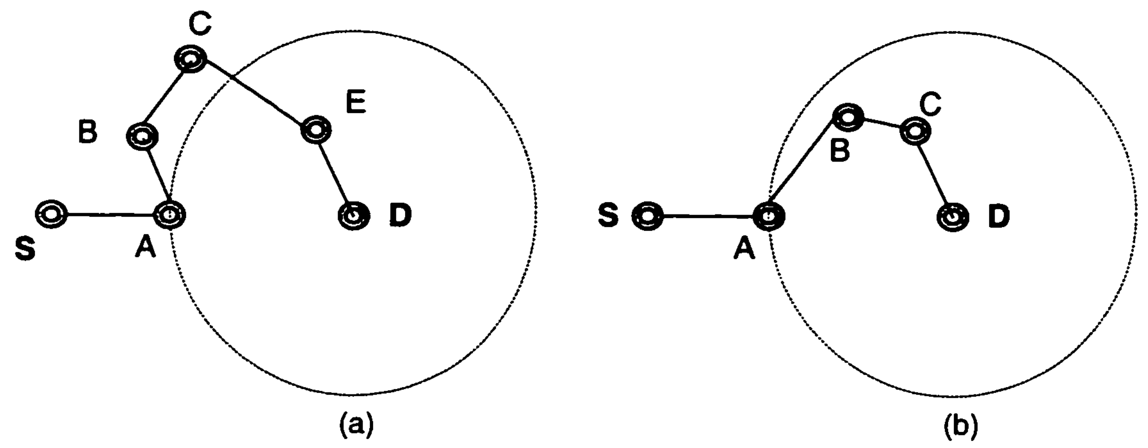


Fig 1.5 In (a), although there is a route from *S* to *D*, the greedy algorithm will fail because *A* will forward the message to *B*, which will forward it back to *A*. In (b), the algorithm will succeed in finding a route.

1.2.3 Perimeter Routing

Perimeter routing [3] is a distributed algorithm for routing that doesn't need duplication of packets. The most important part of this protocol is extracting a Gabriel graph from the network, which is modeled as a unit disk graph. It is known that the Gabriel graph of a unit disk graph is connected and planar [3]. Packets are routed only over the faces of the Gabriel graph. Routing follows the right hand rule. Given a node v on a face f, the boundary of f is traversed in the counterclockwise (clockwise if f is the outer face) direction, unless taking an edge would requiring crossing the segment (v_{src}, v_{dst}). In this case, we switch to the next face. Fig 1.6 is an example.

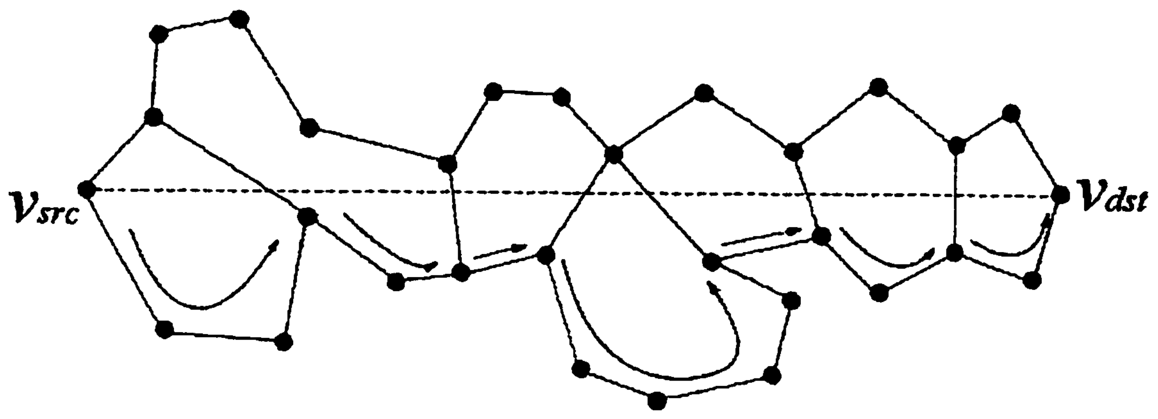


Fig 1.6 Perimeter routing from v_{src} to v_{dst}.

1.2.4 Greedy Perimeter Stateless Routing for wireless network (GPSR)

This protocol is a combination of greedy routing and perimeter routing. The greedy algorithm cannot guarantee packet delivery; in fact, as seen in Chapter 4, its failure rate is pretty high though it usually generates a short path. Perimeter routing never fails, but its performance is not good and usually it generates a long routing path. GPSR is a combination of the two that attempts to use the good qualities of both protocols, while eliminating the disadvantages of both.

GPSR uses greedy forwarding at first. When the greedy algorithm fails, that is, a node finds none of its neighbor nodes is closer to the destination than itself, GPSR switches to perimeter routing. It extracts a planar subgraph and follows the right-hand rule. Once the packet reaches a node closer than where greedy forwarding previously failed for that packet, the packet switches back to greedy forwarding to the destination.

1.3 Organization of the report

In Chapter 2, we describe the RPBR algorithm [1] in detail. This algorithm works in the presence of irregular transmission ranges. Chapter 3 describes the requirements and design of the simulator we have implemented to simulate algorithms for routing in ad hoc networks with irregular transmission ranges. Chapter 4 gives the results of our simulations and Chapter 5 gives our conclusions.

2. Robust Position-Based Routing in wireless Ad Hoc Networks with Irregular Transmission Ranges

This chapter explains the RPBR algorithm in detail. For the correctness proof, see [1].

2.1 Problems caused by irregular transmission ranges

In the previous chapter, we discussed some position-based routing algorithms; all of them assume that each mobile host has a transmission range R, as shown in Figure 1.1 in Chapter 1. However, in reality, this is not always true. Transmission ranges can be affected by various reasons such as obstacles or perturbations. Figure 1.2 shows the concept of irregular ranges. We see that the area a mobile host can reach is not necessarily a disk and the range can vary in a non-uniform manner between $r = (1-\epsilon) R$ and R, $\epsilon > 0$ [1]. Two mobile hosts at distance $d > R$ are not able to communicate directly; at distance $d \leq r$, they are surely able to communicate directly; at distance $r < d \leq R$, they may or may not be able to communicate directly.

The irregularity may create some unidirectional communication links. Moreover, a small variation in the transmission ranges can make the extraction of the Gabriel graph fail. See Fig 2.1 for an example. Assume link uv and uw are bi-directional communication links, and w is in the disk $D_{u,v}$. The distance $d(u, v)$ is between r and R, therefore, v may or may not be able to reach w. If v can reach w, then both v and u agree that the edge $\{u,v\}$ should be removed in the corresponding Gabriel graph. If v cannot reach w, according to v, the edge $\{u, v\}$ should be retained because there is no node in the disk $D_{u,v}$; according to u, the edge $\{u, v\}$ should be removed because there is a node w in the disk $D_{u,v}$. Removing the edge $\{u, v\}$ may disconnect the network, but keeping the edge may cause the graph to be non-planar. Thus perimeter routing will not be consistent and might fail.

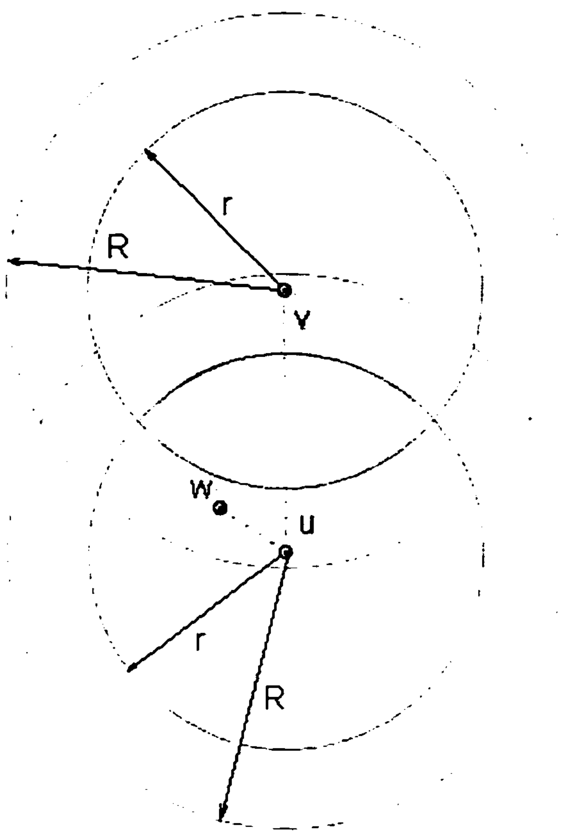


Fig 2.1 Hosts *u* and *v* have inconsistent views on whether to retain the edge *{u, v}* in the Gabriel Graph or not.

RPBR is a distributed protocol that ensures message delivery in a connected network whenever the ratio of the maximum transmission range to the minimum transmission range is at most $\sqrt{2}$ [1]. This restriction guarantees that for any edge *{u, v}*, if there is another host *w* in the disk of this edge, then at least one of *u* and *v* knows the existence of *w*.

2.2 The Routing scheme

Before we explain the details of the routing scheme, we make the following assumptions.

Assumptions:

Assume a set of mobile hosts spread out in a Euclidean plane; each mobile host knows its own position *(x, y)*. The minimum transmission range is r and the maximum transmission range is R. Any two hosts at distance $d \leq r$ can communicate directly, at distance $d > R$ cannot communicate directly, at distance $r < d \leq R$ may or may not be able to communicate directly. Bidirectional communication is enforced by requiring that a communication link is valid only when an acknowledgement is received.

2.2.1 Overview

This routing protocol consists of three phases, the completion phase, the extraction phase and the routing phase. The completion phase will add virtual edges to the graph G and get a super-graph $S(G)$ of G. This phase can guarantee that the Gabriel graph extracted from $S(G)$ is planar and connected. Once the completion phase is done, the extraction phase will extract a connected and planar spanning subgraph from $S(G)$. Message delivery is done at the routing phase. There is no central controller for all the computations and all phases are executed locally at hosts.

2.2.2 The completion phase

Each host u establishes a adjacency list $L(u)$ and broadcasts a *neighbor discover* request. Any host v receiving such a request immediately responds to u with its own coordinates.

If u receives the response, it adds v into $L(u)$. This neighbor discovery process guarantees bidirectional communication links. All the nodes V and the edges E between a host and its neighbors form the graph G. Every node added to $L(u)$ is marked unprocessed. For every unprocessed v, u runs the processing function as follows.

Processing of $\{u, v\}$ by u

For every $w \in L(u)$, if $w \in D_{u,v}$ and $w \notin B(v, r)$ (as in Fig 2.1) then

- u sends the message "(new, w)" to v, together with the coordinates of w;
- u sends the message "(new, v)" to w, together with the coordinates of v;
- u marks v processed.

End-Processing

While processing, every node is ready to receive new node messages from its neighbors and to update its adjacency list L.

Updating the adjacency list of u

When node u receives a message "(new, w)" from neighbor v, u checks whether $w \in L(u)$.

If it is already there, then do nothing.

If not, then u adds w to $L(u)$ and marks it unprocessed. The edge $\{u, v\}$ is set as a virtual edge; w is a virtual neighbor of u. [1] Node u stores the path of the virtual edge $ve(u, w) = (u, v, w)$. The length of $ve(u, w)$ is 2 because it consists of two edges. See Fig 2.2.

End-updating

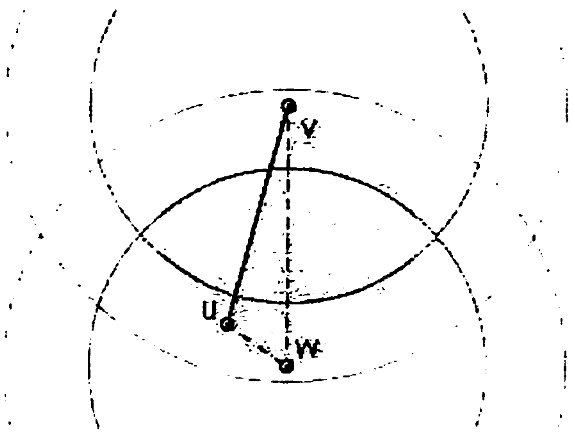


Fig 2.2 Virtual edge *(u, v) = (u, w, v).*

When processing of a virtual neighbor, if an edge is already a virtual edge, it will be embedded in a new virtual edge. See Fig 2.3 and 2.4 for an example.

Theoretically, the length of a virtual edge (the number of edges in the path corresponding to a virtual edge) is not limited. Any virtual edge can only connect nodes at Euclidian distance less than *R*.

Since the processing of edges may induce sending messages to virtual neighbors, a virtual routing protocol is required. The following sending and forwarding functions are in the virtual routing protocol [1].

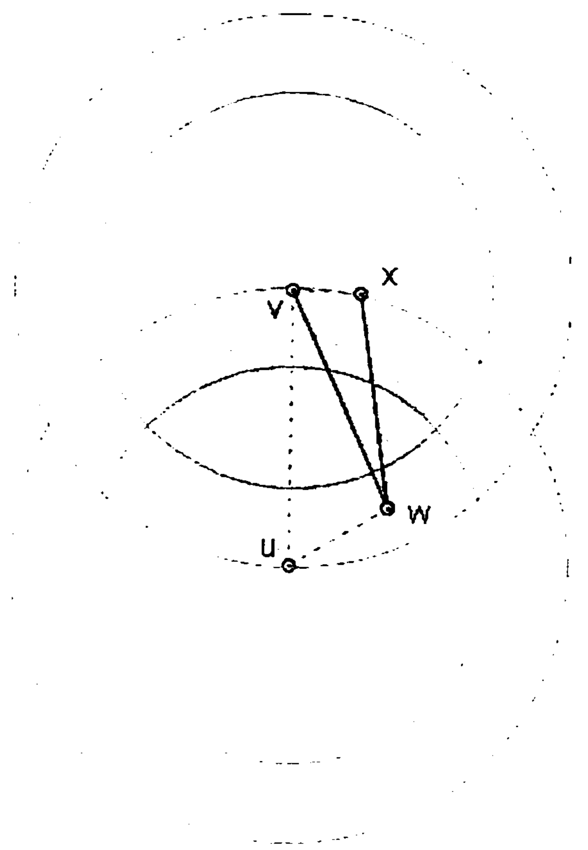
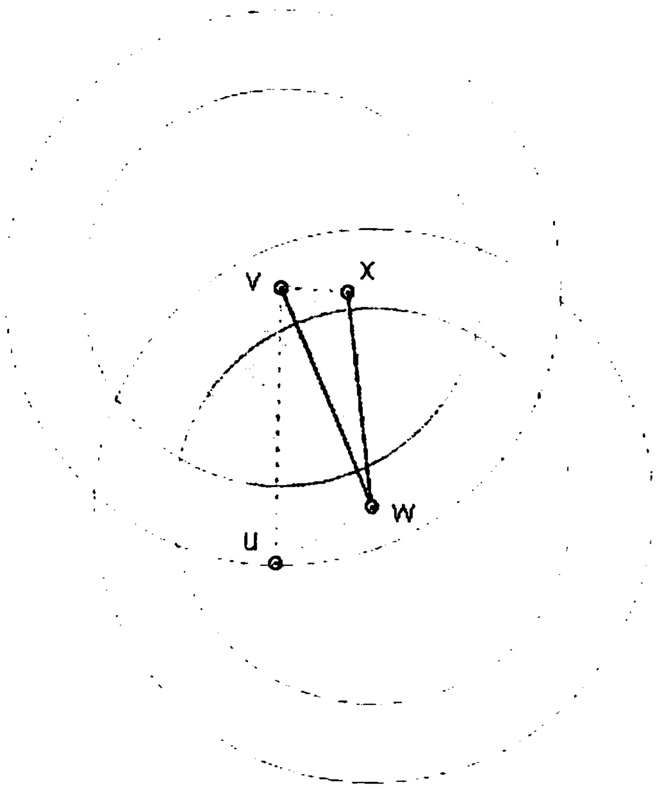


Fig 2.3 *{v, u}*, *{u, w}* and *{v, x}* are direct communication links; *{v, w}* is a virtual edge. *ve(v, w) = {v, u, w}*. The length of *ve(v, w)* is 2.

Fig 2.4 *{x, w}* is a virtual edge, *ve(x, w) = {x, v, w}*. *{v, w}* is also a virtual edge, therefore , *ve(x, w) = {x, v, u, w}*. The length of *ve(x, w)* is 3.



Sending a message

If node *u* wants to send a message *M* to a neighbor *v* (*v* could be virtual neighbor), *u* calls *send(v, M)*

- if *v* is a direct neighbor, the message *(v, M)* is transmitted to *v*;
- if *v* is a virtual neighbor, $ve(u,v) = (u, w, v)$, *u* calls *send(w, (v, M))*.

End-sending

Note: if *w* is a virtual neighbor of *u*, $ve(w, u) = (w, x, u)$, *u* calls *send(x, (w, (v, M)))*. *u* repeats the process and eventually, the message will be sent through a path consisting of direct connected edges.

Forwarding a message

When node *u* receives a message *(u, M)* from a neighbor,

- if $M = (v, M')$, $v \neq u$, then u calls *send(v, M')*;
- otherwise *u* stores *M*;

End-forwarding

When a node marks all nodes in its adjacency list *L* processed, it has completed the completion phase and enters the extraction phase. Note: A node *u* may find a virtual neighbor *v* after entering the extraction phase. In this case, it stops the extraction phase and returns to the completion phase to process the edge $\{u, v\}$. Switching between the completion phase and the extraction phase does not affect the correctness of this protocol [1].

At the end of the completion phase, *S(G)* is generated. Every node knows the coordinates of its neighbors (direct or virtual) and the actual paths corresponding to virtual edges.

2.2.3 The extraction phase

In this phase, the Gabriel graph of *S(G)*, denoted by *GG(S(G))*, is extracted. The vertex set of *GG(S(G))* is *V*, the set of nodes. The edge set, includes those edges in *S(G)* that satisfy the Gabriel graph condition described before. The following is the function to validate edges.

Validating {u, v} by u

u checks every node *v* in *L(u)*,

- if there exists $w \in L(u) \cap D_{u,v}$, *u* deletes *{u, v}*
- otherwise the edge *{u, v}* is kept in *GG(S(G))*.

End-validation

After processing all nodes in *L(u)*, it has completed the extraction phase and is ready to enter the routing phase. As in the extraction phase, if any node is aware of a new neighbor, it returns to the completion phase, then runs the extraction phase and finally comes back to the routing phase. This switching between phases does not affect the correctness of this protocol [1].

2.2.4 The routing phase

Routing in *GG(S(G))* is performed according to the strategy of GPSR which combines both the greedy algorithm in *G* and perimeter routing in *GG(S(G))*. When greedy routing fails, it switches to perimeter routing and when greedy routing is applicable, it switches back. Also, perimeter routing in *GG(S(G))* can be applied alone. Note that perimeter routing may route message through virtual edges, in such cases, the virtual routing protocol is used.

3. Protocol Simulation

In this chapter, we describe the requirements and design for our simulator for routing algorithms. The simulator allows for two kinds of simulations: The GUI simulation for studying and analyzing the routing protocols and the console simulation for running several simulations with the purpose of collecting statistics. Both of them are developed in JAVA.

3.1 Requirement and specification

In the GUI simulation, the following functions will be implemented:

- Generate new simulation
- Show/hide direct neighbor links
- Show/hide ranges of nodes
- Show/hide virtual edges
- Show/hide the Gabriel graph
- Move nodes
- Select source and destination nodes
- Draw a disk $D_{u,v}$
- Show actual path corresponding to a virtual edge
- Show/hide Dijkstra's shortest path
- Show/hide the route found by RPBR
- Show/hide the route found by the greedy algorithm
- Show/hide the route found by perimeter routing

- Show/hide the route found by GPSR
- Modify inner range *r* of a node, *R/r* ratio, node density, and probability of being a direct neighbor.

In the console simulation, users will be asked to input the following:

- Number of run times
- Width of the area
- Length of the area
- Density of node
- Inner range of node and *R/r* ratio
- Probability of being direct neighbor
- Output file name

In the output file, besides the above parameters, the following data is provided for each run:

- Length of the shortest path as discovered by Dijkstra's algorithm
- Length of the route found by RPBR
- Length of the route found by the greedy algorithm
- Length of the route found by perimeter routing
- Length of the route found by GPSR
- Average length of virtual edges
- Distribution of lengths of virtual edges
- Percentage of virtual edges in the extracted Gabriel graph

- Percentage of virtual edges in routes found by RPBR

For statistical purposes, we collect the average over several runs:

- Normalized length of routes found by all algorithms
- Failure rate of all algorithms
- Average length of virtual edges of all runs
- Distribution of lengths of virtual edges over all runs
- Percentage of virtual edges in the extracted Gabriel graph of all runs
- Percentage of virtual edges in routes found by RPBR of all runs

3.1.1 Hardware requirement

Minimum Requirements:

- PC's with at least 266 MHz Processors, 32 MB RAM
- Since the console simulation will output results to a file, extra disk space is needed.

Optimal Performance:

- We recommend a PIII processor with 1 GHz and 128 MB RAM.

3.1.2 Software Requirements

- Microsoft Windows 9x/XP/NT/2000, Linux or UNIX.
- Java runtime environment 1.3 and above

3.2 Use case diagrams

In this section, we show the use cases for the GUI simulation (Figure 3.1) as well as the console simulation (Figure 3.2).

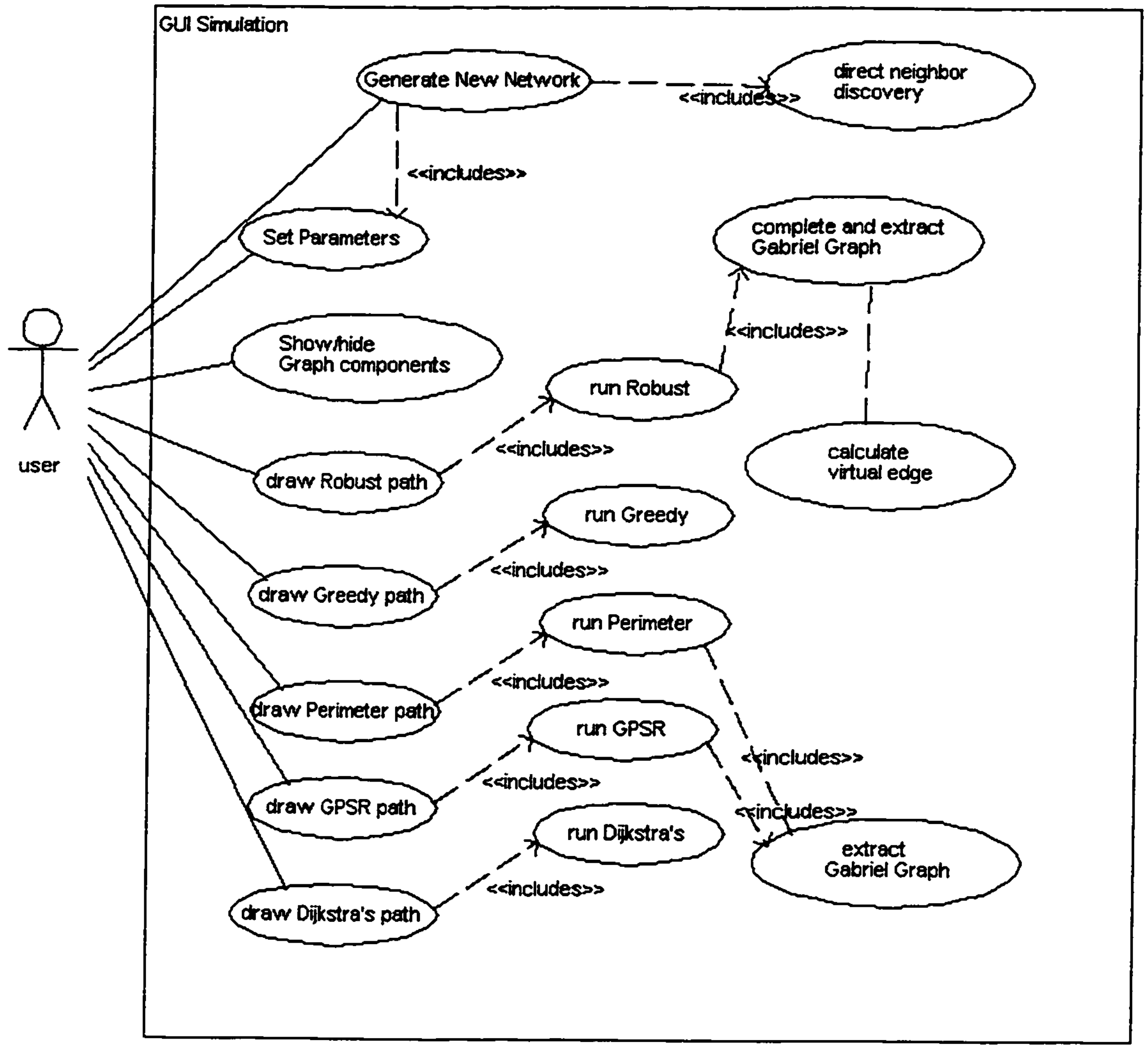


Fig 3.1 The GUI simulation use case.

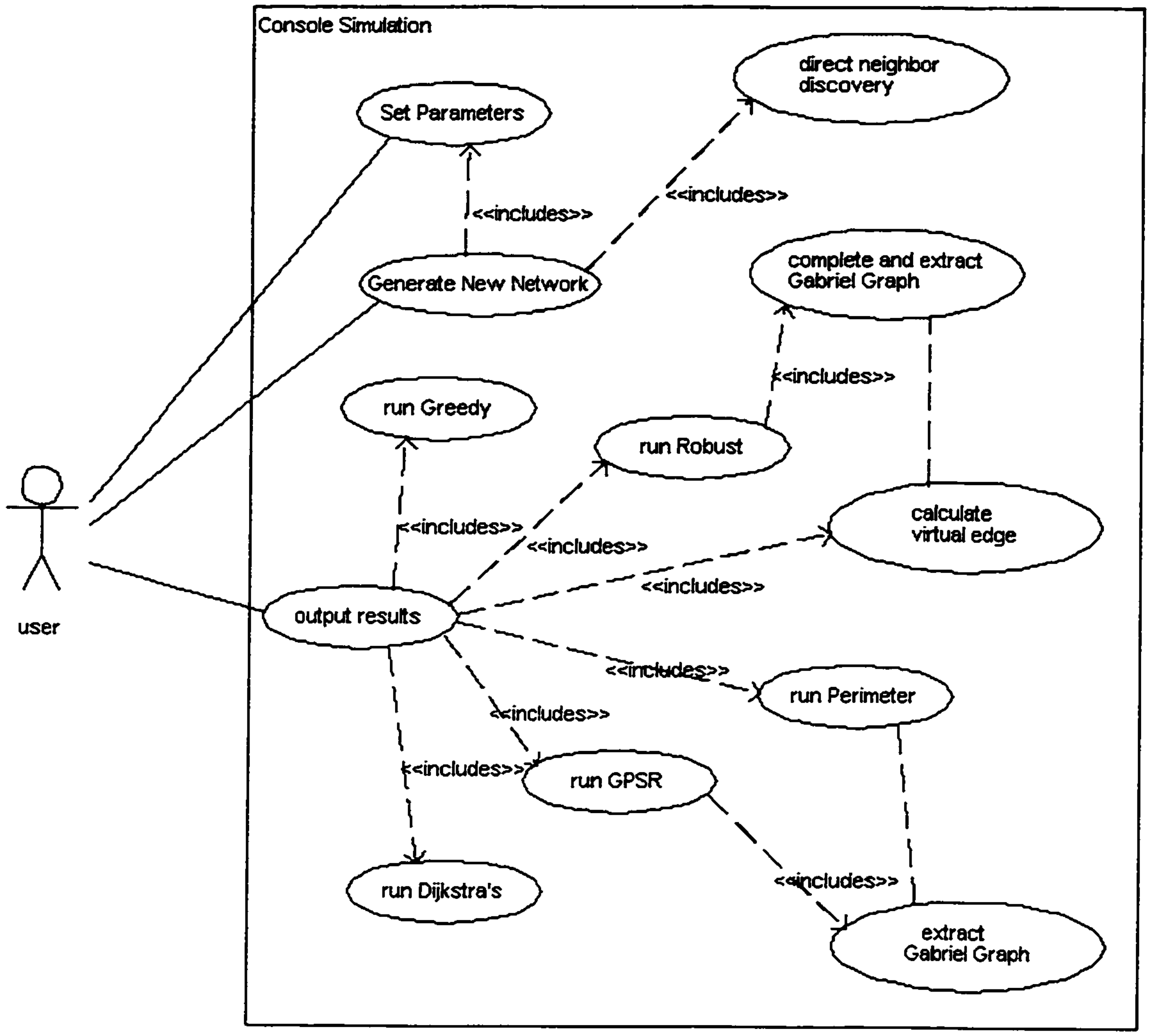


Fig 3.2 The console simulation use case.

3.3 Design

The class Simulation generates and controls the user interface, and contains the main( ) function. In the GUI version, it generates all components in the interface such as buttons, text fields and drawing area. It also includes the initialization function that randomly generates nodes. The drawing area in the GUI interface is realized by the class DrawCanvas, which represents all the drawing components such as nodes, edges, routes and so on.

For the unit disk graph *G(V,E)*, we define the class Node for *V* and the class Edge for *E*. Position is a simple class to represent coordinates of nodes or points. Local calculations such as neighbor discovery, distance computation, finding the next edge according to the right hand rule, and the processing and extracting functions in RPBR are implemented in the class Node. In the GUI version, the class Node also draws the nodes, neighbor links, the Gabriel graph and ranges in the drawing area. The class Edge can instantiate both normal edges and virtual edges. It can calculate the length of virtual edges. In the GUI version, it draws disks, edges and virtual edges in the drawing area.

For routing algorithms, we declare five classes corresponding to the five protocols. The routing algorithms are implemented in these five classes. They are Dijkstra, Robust, Greedy, Perimeter and GPSR. Each algorithm class can run the corresponding routing algorithm to find the route and calculate the length. In the GUI version they also draw the routes.

Class diagrams

In this section, we give the overall class diagrams for both the GUI simulation and the console simulation.

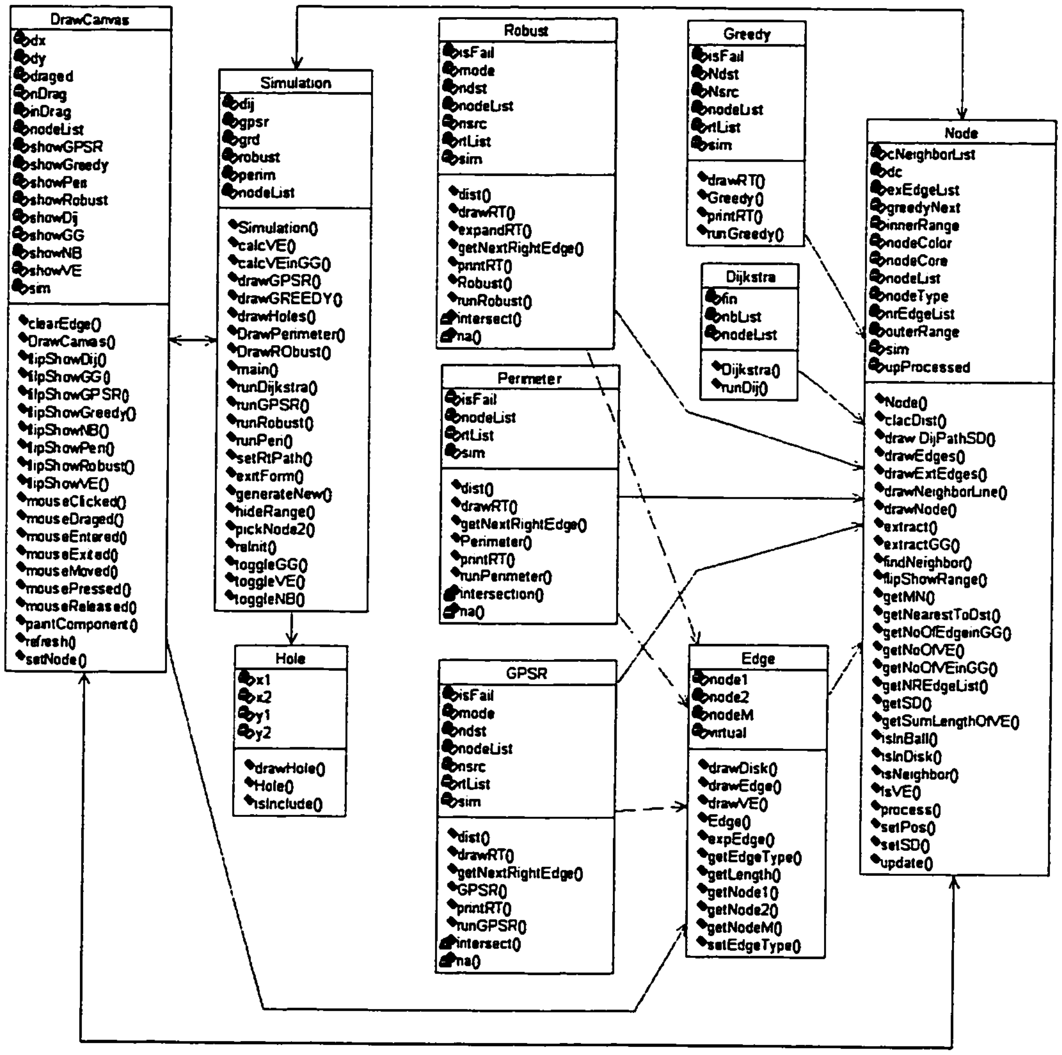


Fig 3.3 The GUI simulation class diagram.

Figure 3.3 is the class diagram for the GUI simulation, showing the relationship between the different classes.

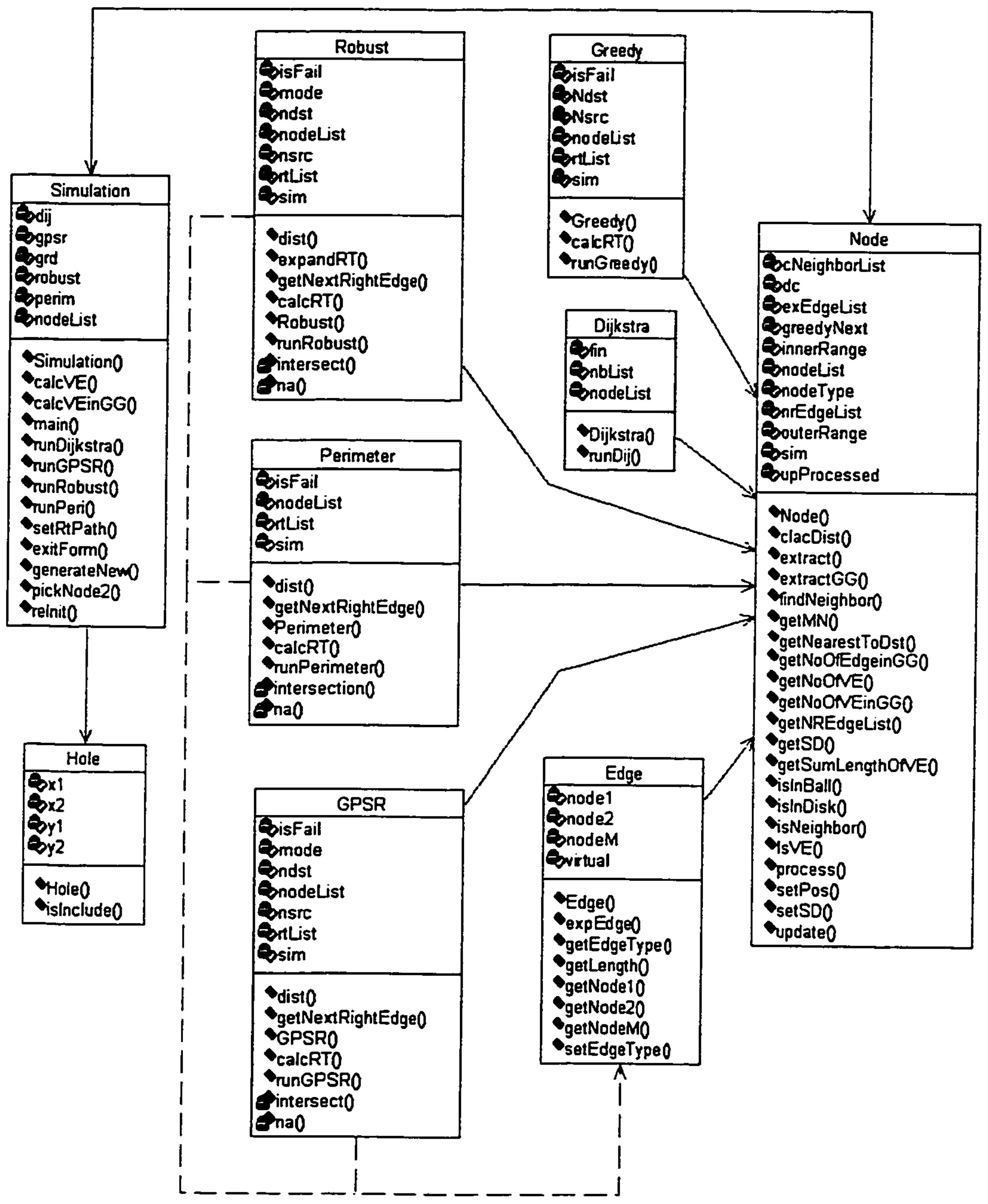


Fig 3.4 The console simulation class diagram.

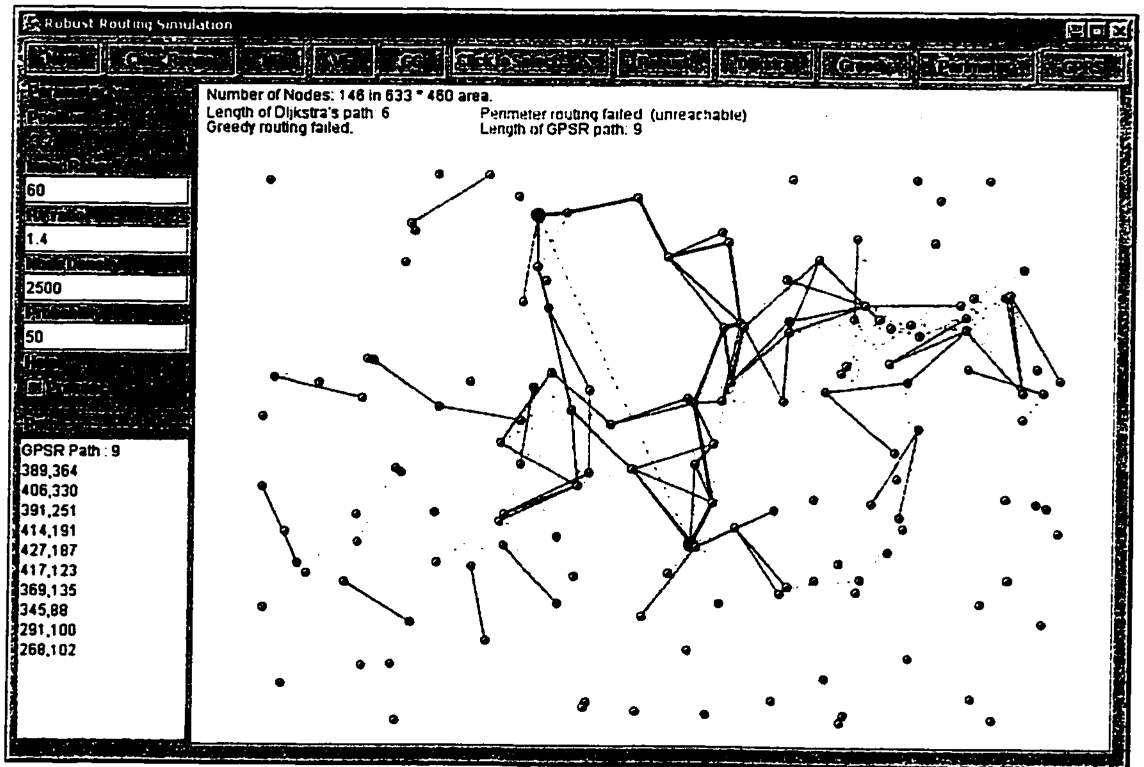
Figure 3.4 is the class diagram for the console simulation, showing the relationship between all the classes.

3.4 GUI simulation

Fig 3.5 shows a screen shot of the GUI. All the function buttons are on the top. Initial parameters and results of route paths are on the left. It lists all the nodes in the route in order. It also shows the number of virtual edges, average length of virtual edges, percentage of virtual edges in the Gabriel graph and percentage of virtual edges in the RPBR route.

In the middle is the drawing area in which the ad hoc network model is shown.

In the top left of the drawing area, the area of the plane and the number of mobile nodes are displayed. The lengths of routes found by the different routing algorithms are also displayed. By clicking on a routing algorithm, the route found by the algorithm will be displayed. The routes found by different algorithms are shown in different colors.



Number of Nodes: 146 in 633 * 460 area.
Length of Dijkstra's path 6
Greedy routing failed.
Perimeter routing failed (unreachable)
Length of GPSR path: 9

60
1.4
2500
50

GPSR Path : 9
389,364
406,330
391,251
414,191
427,187
417,123
369,135
345,88
291,100
268,102

Fig 3.5 Screen shot of the GUI.

The following sections give more details about the GUI, including the drawing components, the parameters and the functions.

Drawing components:

Node: Small orange circles are nodes, the red one is the source and the blue one is the destination.

Range: 2 rings represent the inner and outer range of a node. The outer range is light gray; the inner range is cyan.

Disk: Light gray disk with the length of an edge as diameter.

Direct neighbor link: Light gray dotted lines represent direct communication links.

Virtual edge: Magenta edges are virtual edges. Black dotted line shows the actual path of the virtual edge.

Gabriel graph: Yellow thick lines represent the edges of the Gabriel graph.

Dijkstra's path: Dijkstra routes are shown in blue.

RPBR route: RPBR routes are shown in orange.

Perimeter route: Perimeter routes are shown in red.

Greedy route: Greedy routes are shown in black.

GPSR route: GPSR routes are shown in dark gray.

Parameters:

Position: Current mouse position. This can help to estimate the position of a node.

Inner Range: The inner transmission range of a node.

R/r ratio: The ratio *R/r* = the outer range / the inner range. The value is from 1 to $\sqrt{2}$.

Node density: Number of nodes = area of plane / node density.

Probability: The probability of whether a node between inner and outer range can be reached directly.

To bring a change of parameters into effect, a new model must be created by clicking the "New" button.

Functions:

New: Randomly generate new wireless ad hoc network model according to the initial parameters.

Clear Range: Clears node ranges, disk, and actual path of virtual edges.

NB: Show/hide direct neighbor links.

VE: Show/hide virtual edges.

GG: Show/hide the Gabriel graph.

Click to select: Select source/destination node, select 2 nodes of an edge to draw a disk, select 2 nodes of a virtual edge to draw actual path.

Dijkstra: Show/hide the route discovered by Dijkstra's algorithm.

Robust: Show/hide the route found by RPBR.

Greedy: Show/hide the route found by Greedy routing.

Perimeter: Show/hide the route found by perimeter routing.

GPSR: Show/hide the route found by GPSR.

All routes will also be represented by a list of coordinates of nodes in the left text field.

3.5 Console simulation

In the console simulation, the user can specify the number of times the simulation is to be run, the width and length of the area, node density, inner transmission range, *R/r* ratio, probability to be a neighbor when distance of two nodes is between *R* and *r*, and the name of the output file.

In the output file, it confirms all the parameters specified by the user: the normalized lengths of routes found by all algorithms, average virtual edge length, distribution of lengths of virtual edges, percentage of virtual edges in the Gabriel graph, percentage of virtual edges in the route found by RPBR, and failure rate for each algorithm.

Note that "-1" means routing failure. We ignore cases where length of Dijkstra' path is -1, which means the destination node is not reachable by the source node.

In the end of the result file, we can see the average normalized length of each routing protocol and its failure rate; we can also see the average virtual edge length, distribution of lengths of virtual edges, percentage of virtual edges in the Gabriel graph and percentage of virtual edges in the route found by RPBR.

4. Simulation Results and Analysis

In this chapter, we describe the results of our simulations of the four algorithms: Greedy routing[13], perimeter routing[3], GPSR[10] and RPBR[1]. They are all compared with Dijkstra's shortest path algorithm[4]. We study the failure rates of the algorithms, and the lengths of routes produced by them. For RPBR, we also study the average length of virtual edges, the percentage of virtual edges in the Gabriel graph produced by RPBR and the percentage of virtual edges in the route found by RPBR.

In all simulations, we use a fixed sized area of which the width and length are both 1000 units. For each scenario, we run the simulation 1000 times. We use a fixed inner range of 60 units. Other parameters are changed in order to observe the effects.

Note that Dijkstra's algorithm is implemented here strictly for comparison; since it is not distributed and local, it cannot be used as a routing algorithm in ad hoc networks. In our experiments, we consider an algorithm to have failed when it doesn't terminate even though the number of hops in the route being constructed by the algorithm exceeds $n \log n$ where n is the number of nodes in the network. Recall that the greedy algorithm fails when all of a node's neighbors are further from the destination than the node itself. Perimeter routing may fail if the graph being routed on is disconnected or non-planar. We also note that the only circumstance in which Dijkstra's algorithm fails is when the graph is not connected; we ignore such cases in our description of the results on failure rates.

To make the results on lengths of routes found by the different algorithms more meaningful, we normalize the length of the path found by each algorithm to that discovered by Dijkstra's algorithm for the same case. Since perimeter routing and GPSR may fail in some cases where RPBR succeeds, to provide a fair comparison, we only consider the cases when both perimeter routing and GPSR succeed. However, we do consider cases in which the greedy algorithm fails, because otherwise the performance of GPSR, Greedy and RPBR would be identical.

4.1 Results

In this section, we analyze the effects of density, R/r ratio and $P_{neighbor}$ on the performance of the algorithms. We say that the density is $1/x$, when there is 1 node in each x square units on average. Therefore, as x increases, the density decreases. We also define $P_{neighbor}$ to be the probability that for an arbitrary node v in the network, a node at distance between r and R from v is a neighbor of v.

4.1.1 Effects of density of nodes

In this section, we study the effect of the density of nodes on the performance of the algorithms. We fix the R/r ratio to be 1.4 and $P_{neighbor}$ to be 0.5 in these experiments.

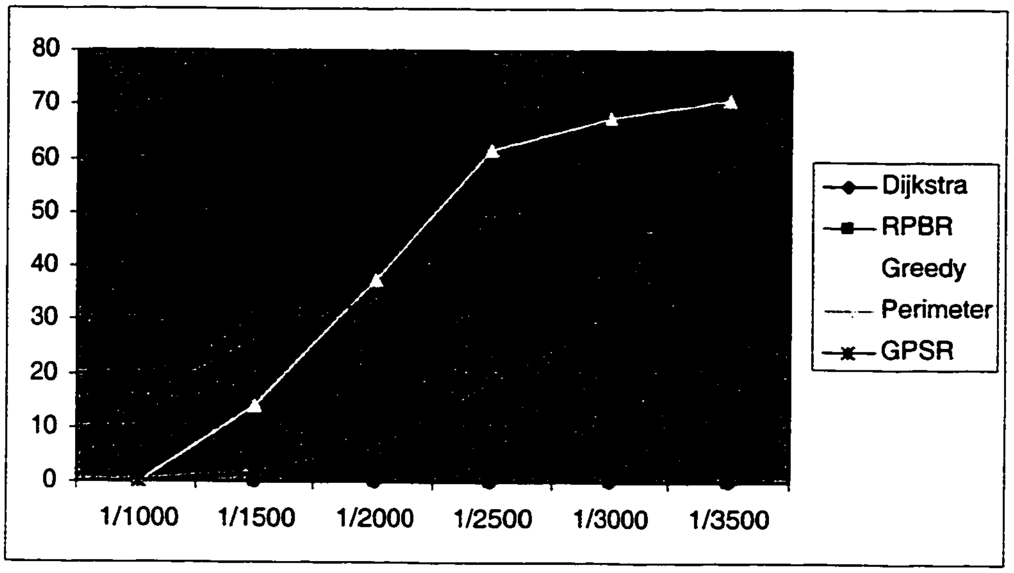


Fig 4.1 Effect of density of nodes on failure rates.

Figure 4.1 shows the effect of density on the failure rates of the algorithms. The failure rates increase along with the density decreasing. We can see the greedy algorithm fails more than others. The failure rate of perimeter routing is also high because irregular transmission ranges cause the Gabriel graph to be non-planar or disconnected. GPSR combines greedy and perimeter routing; when the greedy algorithm fails, it switches to perimeter routing. Recall that perimeter routing can fail when transmission ranges are irregular, so GPSR can fail too, but its failure rate is better than perimeter routing. As expected, RPBR never fails.

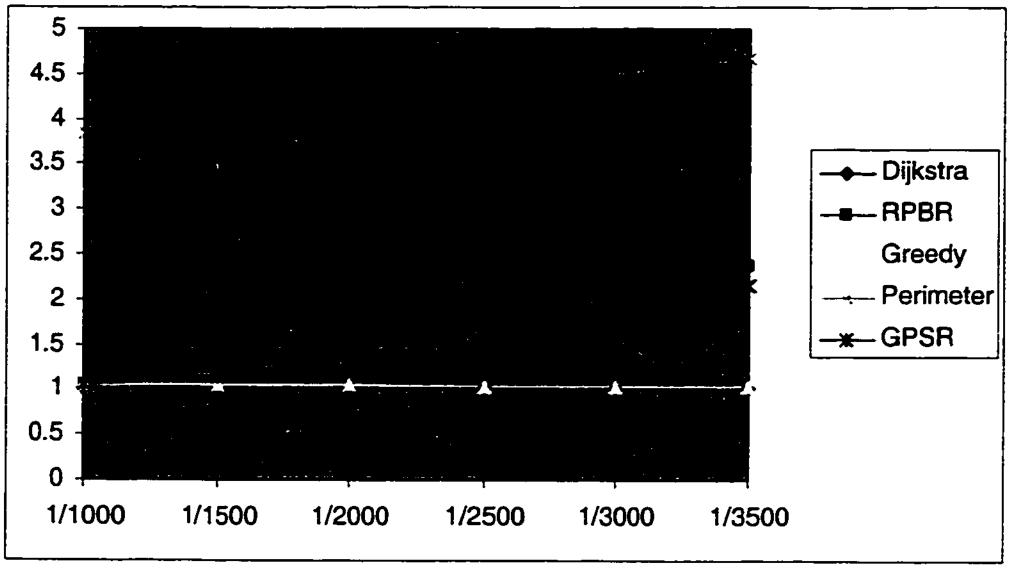


Fig 4.2 Effect of density of nodes on lengths of routes.

Figure 4.2 shows the effect of density on the length of routes produced by the algorithms. Generally, when the density decreases, the length of routes found by the algorithms increases. This is because when the network is less dense, it is more possible to detour to find a route. However, for perimeter routing, when the network is too dense, it will cover more edges hence more hops. Greedy is the best in position-based protocols; in fact, its performance is almost identical to Dijkstra's algorithm, but this is only because the cases where it fails have been ignored. Perimeter routing produces the longest routes. RPBR and GPSR combine the advantages of the greedy algorithm and perimeter routing, therefore they are in the middle. The average length of the routes generated by RPBR is very close to that of the routes produced by GPSR. It is a little bit higher because it uses virtual edges, which have average length two. Since virtual edges only comprise a very small percentage in RPBR routes, they only increase the length a little.

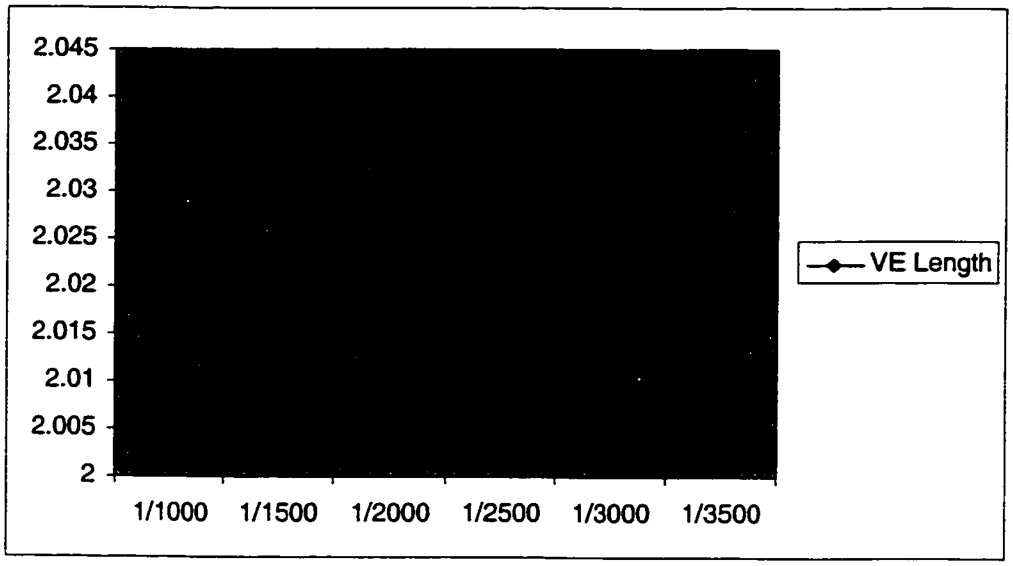
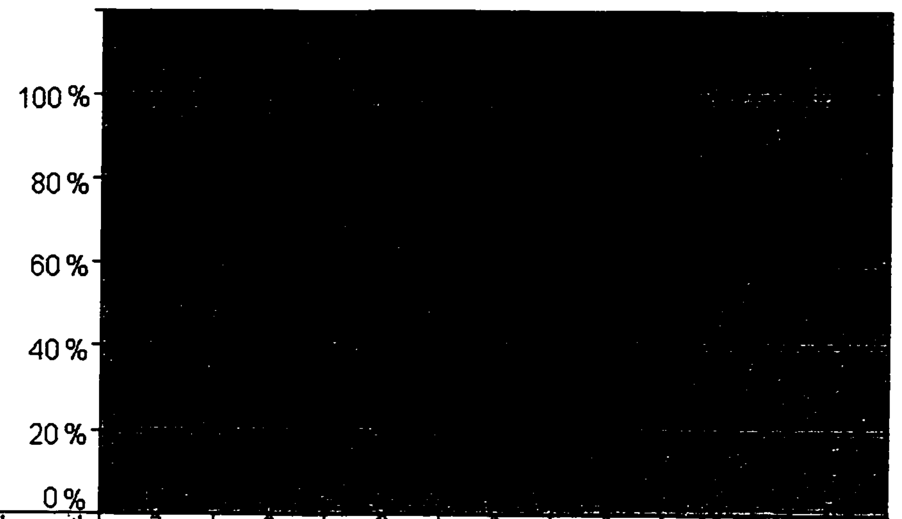


Fig 4.3 Effect of density of nodes on the average length of virtual edges.

Figure 4.3 shows the effect of density on the average length of virtual edges. The average length of a virtual edge is very close to 2 regardless of the density of the nodes. The average virtual edge length is a little higher when the network is denser. In a dense network, the area inside *B(u,R)* but excluding *B(u,r)* contains more nodes that the node cannot see directly, and so there is an increased likelihood of forming virtual edges, as well as virtual edges of length greater than 2.



% of edges with length	1/1000	1/1500	1/2000	1/2500	1/3000	1/3500	1/4000
◆ 2	96.00	96.76	97.28	97.68	98.02	98.28	98.49
■ 3	3.91	3.18	2.69	2.30	1.96	1.70	1.50
4	0.09	0.06	0.04	0.03	0.02	0.02	0.01
* 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Max Length	6	5	5	4	4	4	4

Fig 4.4 Effect of density of nodes on the distribution of lengths of virtual edges.

Figure 4.4 shows the effect of density on the distribution of lengths of virtual edges and the maximum length. Regardless of the density, over 95% of virtual edges have length 2. When the density decreases, the percentage of virtual edges with length 2 increases, meanwhile, the percentage of virtual edges with higher length increases. The maximum length of a virtual edge can be higher than 4 when the density is higher than 1/2500, but the percentage of such long virtual edges is lower than 0.01%. In all our simulations, no virtual edge of length greater than 6 was encountered. The maximum length of virtual edges decreases along with the decrease of density; as expected, long virtual edges are more likely to be formed when the network is denser.

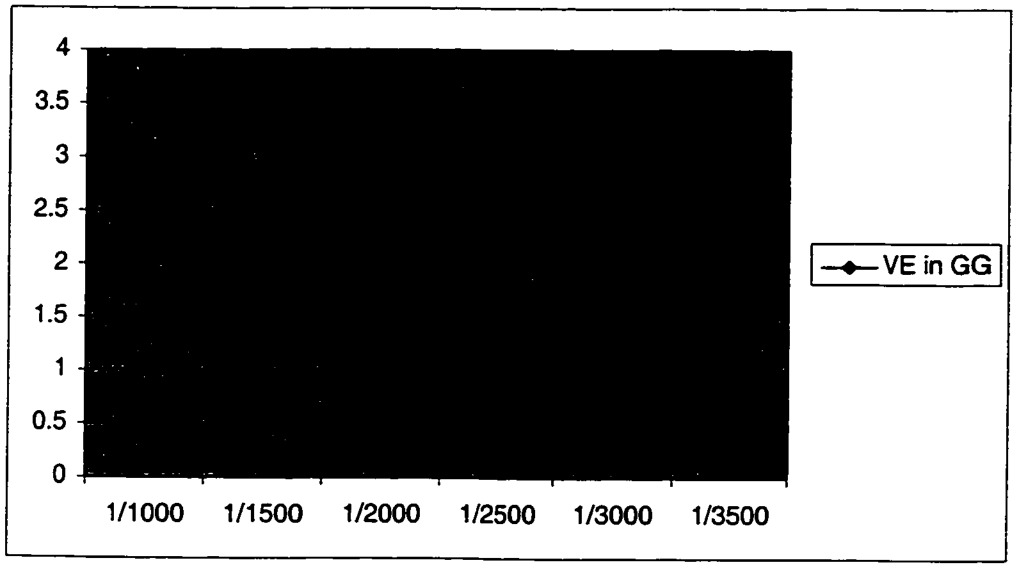


Fig 4.5 Effect of density of nodes on the percentage of virtual edges in GG.

Figure 4.5 shows the effect of density on the percentage of virtual edges in the Gabriel graph. The percentage of virtual edges in the Gabriel graph is higher when the network is less dense. This is interesting because when the density decreases, the total number of virtual edges and the number of edges in the Gabriel graph both decrease. It is hard to predict which decreases faster. The result shows the latter decreases faster.

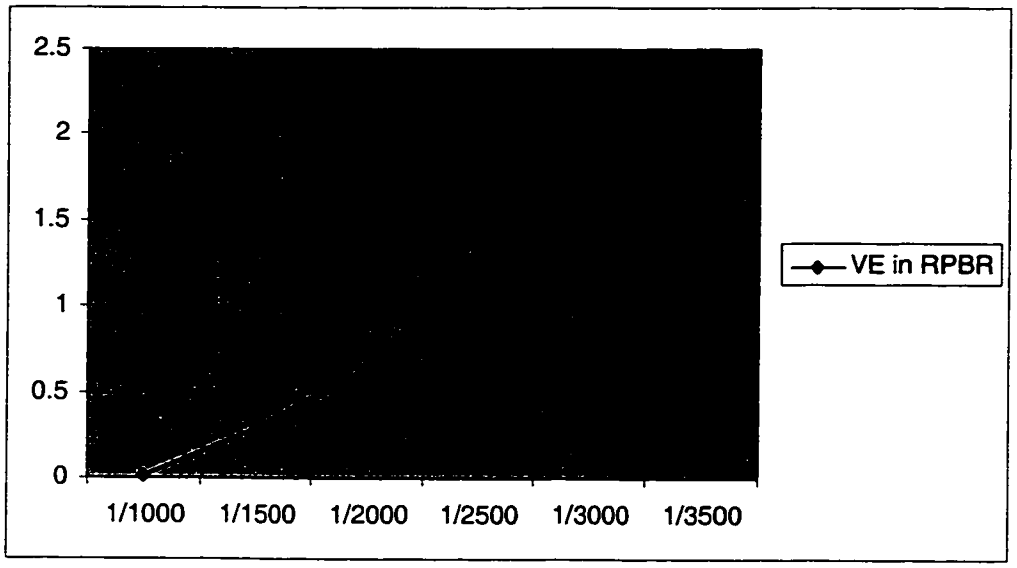


Fig 4.6 Effect of density on the percentage of virtual edges in RPBR routes.

Figure 4.6 shows the effect of density on the percentage of virtual edges in the route produced by RPBR. The percentage of virtual edges in the RPBR route is higher when the network is less dense. This follows the pattern of the previous chart. We notice that the percentage of virtual edges in RPBR is close to 0 when density is 1/1000. This is because in such a dense network, the greedy algorithm hardly fails and so RPBR can find most routes by using the greedy algorithm without switching to perimeter routing.

4.1.2 Effects of *R/r* Ratio

In this section, we study the effect of the *R/r* ratio on the performance of the algorithms. We fix $P_{neighbor}$ to be 0.5, a fixed inner range of 60 units and density to be 1/2500 in these experiments.

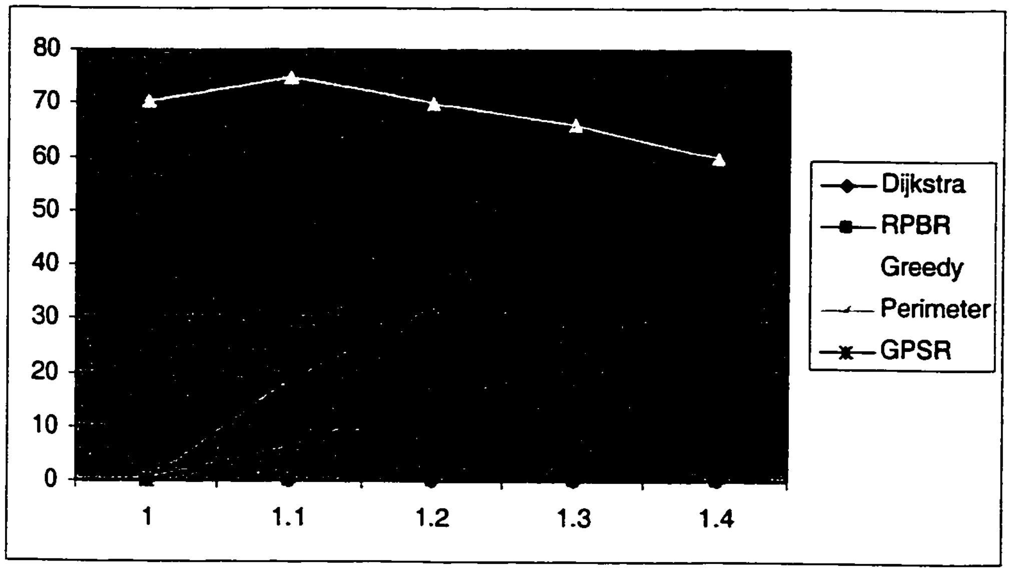


Fig 4.7 Effect of *R/r* ratio on failure rates.

Figure 4.7 shows the effect of the *R/r* ratio on the failure rates of the algorithms. Once again, Dijkstra's algorithm and RPBR never fail. When *R/r* ratio increases, the failure rate of the greedy algorithm decreases a little because nodes can have more neighbors, and can reach a greater distance in one hop. The failure rate of perimeter routing and GPSR increase when *R/r* ratio increases because the number of nodes between *r* and *R* increases thus the irregularity of the transmission range increases.

When $R = r$, the transmission range is regular and then there are no virtual edges, and so the failure rate of perimeter routing and GPSR is 0 as well.

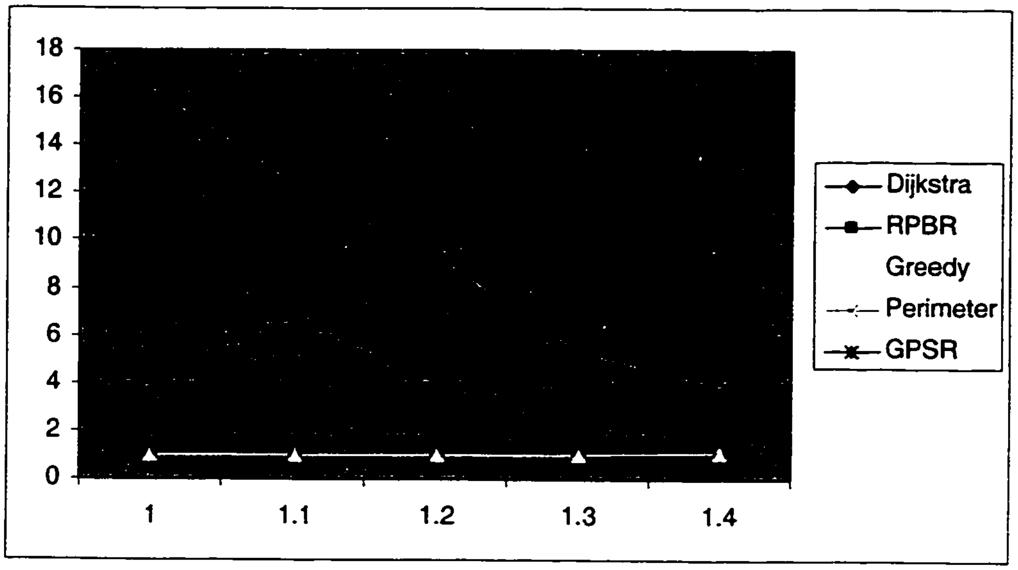


Fig 4.8 Effect of *R/r* ratio on lengths of routes.

Figure 4.8 shows the effect of *R/r* ratio on the length of routes produced by the algorithms. In general, the lengths of routes decrease when the *R/r* ratio increases. This is because nodes can reach further neighbors. Still perimeter routing has the worst performance, greedy routing has the best and GPSR and RPBR are in the middle.

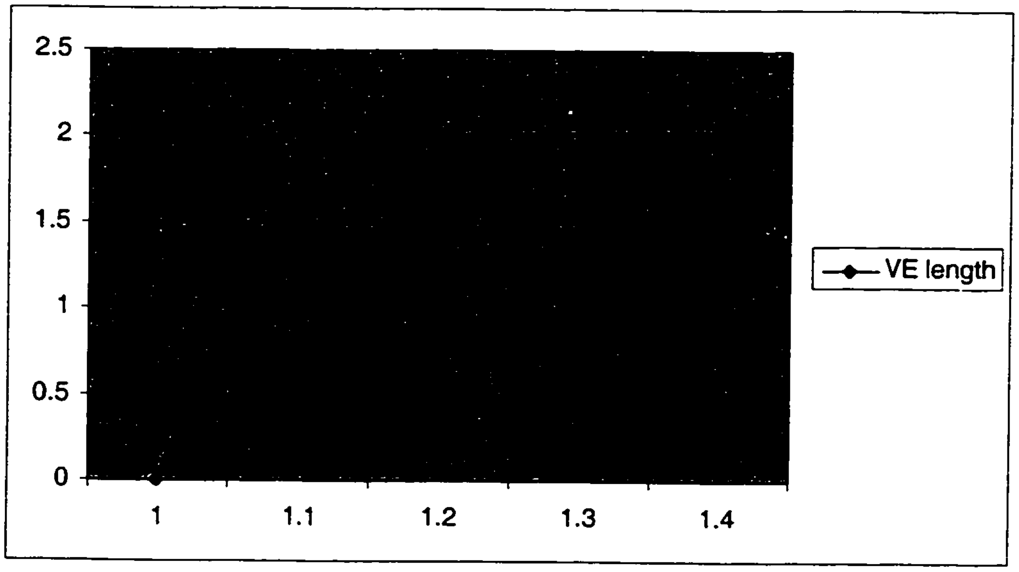
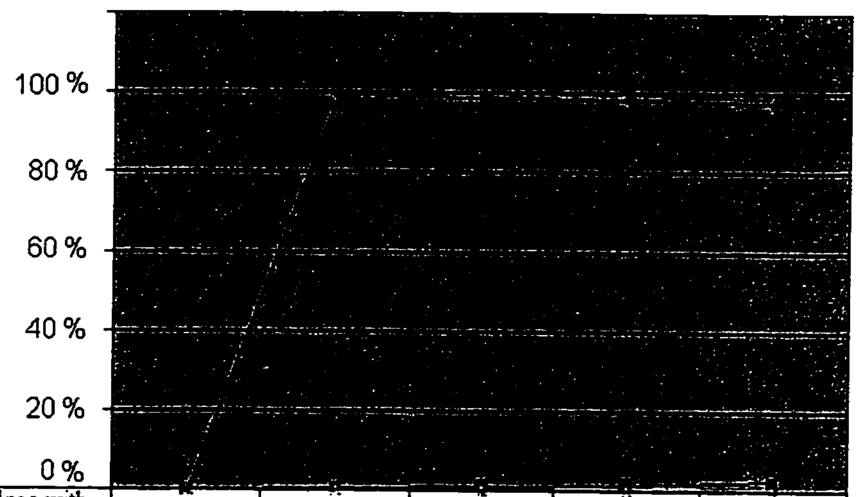


Fig 4.9 Effect of *R/r* ratio on the average length of virtual edges.

Figure 4.9 shows the effect of the *R/r* ratio on the average length of virtual edges. When $R = r$, there are no virtual edges. Along with the increase in the *R/r* ratio, the average length of virtual edges increases too. That means that the increase of irregularity of the transmission range will increase the length of virtual edges by a small amount. However it is always very close to two.



% of edges with length	1	1.1	1.2	1.3	1.4
—◆— 2	0.00	99.70	98.97	98.39	97.72
—■— 3	0.00	0.30	1.02	1.60	2.25
4	0.00	0.00	0.01	0.01	0.03
—✕— 5	0.00	0.00	0.00	0.00	0.00
Max Length	0	3	4	4	4

Fig 4.10 Effect of *R/r* ratio on the distribution of lengths of virtual edges.

Figure 4.10 shows the effect of the *R/r* ratio on the distribution of lengths of virtual edges and the maximum length. The percentage of virtual edges with length 2 is always more than 97%. When the *R/r* ratio increases, the percentage of virtual edges with length 2 decreases, and the percentage of virtual edges with length 3 and 4 increases. The maximum length of virtual edges increases with the increase in *R/r* as well. This means that when the *R/r* ratio is high, longer virtual edges are likelier to be formed.

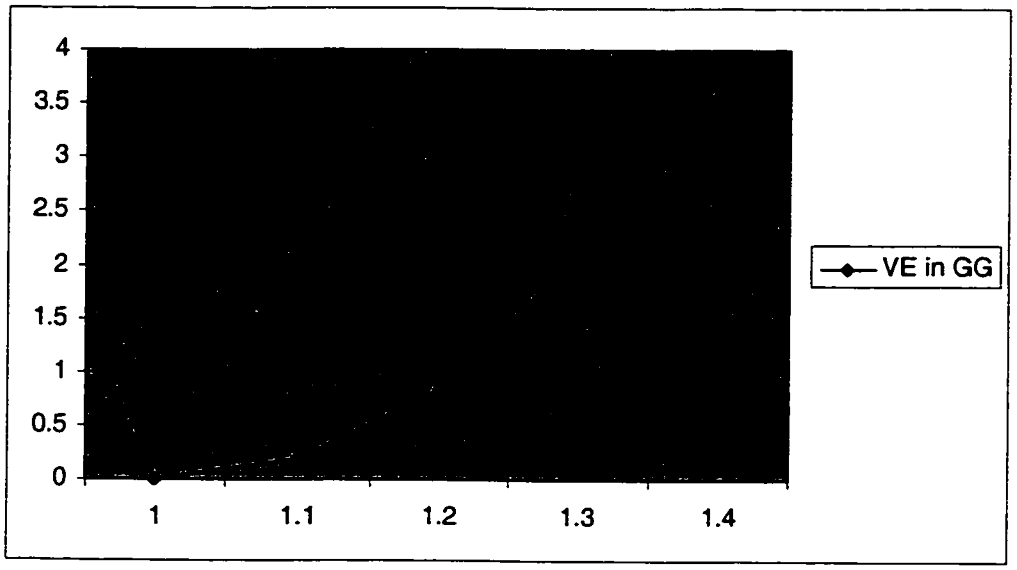


Fig 4.11 Effect of *R/r* ratio on percentage of virtual edges in GG.

Figure 4.11 shows the effect of the *R/r* ratio on the percentage of virtual edges in the Gabriel graph. The percentage of virtual edges in the Gabriel graph increases because there are more virtual edges when the *R/r* ratio increases. Obviously the number of virtual edges increases when the *R/r* ratio increases.

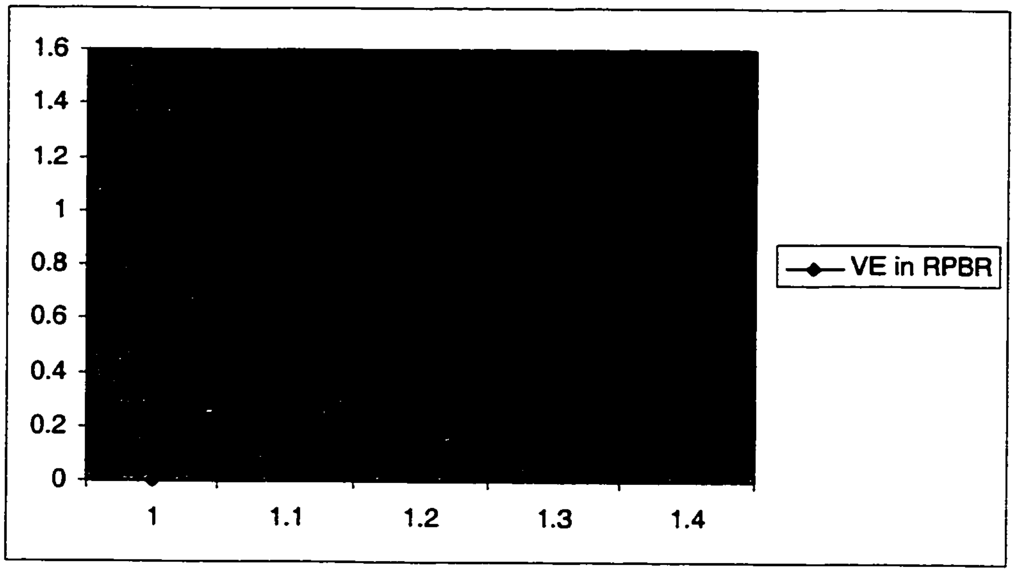


Fig 4.12 Effect of *R/r* ratio on percentage of virtual edges in RPBR routes.

Figure 4.12 shows the effect of the *R/r* ratio on the percentage of virtual edges in routes produced by RPBR. The percentage of virtual edges in RPBR path increases as well for the same reason as in the previous chart.

4.1.3 Effects of $P_{neighbor}$

In this section, we study the effect of $P_{neighbor}$ on the performance of the algorithms. We fix the density to be 1/2500 and *R/r* ratio to be 1.4 in these experiments.

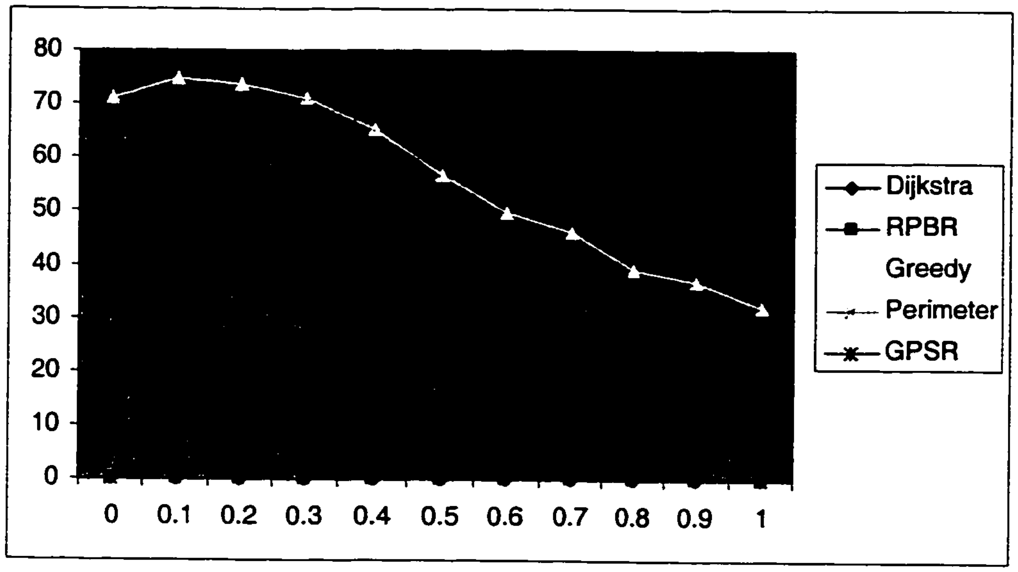


Fig 4.13 Effect of $P_{neighbor}$ on failure rates.

Figure 4.13 shows the effect of $P_{neighbor}$ on the failure rates of the algorithms. When $P_{neighbor}$ is 0, this means that the graph is a unit disk graph with radius *r*, and when it is 1, we have a unit disk graph with radius *R*. In both cases, the transmission range is regular. So, perimeter routing and GPSR don't fail. Generally, an increase in the value of $P_{neighbor}$ decreases the failure rate because a higher value of $P_{neighbor}$ means more nodes in the uncertain range can become direct neighbors. Again, RPBR never fails.

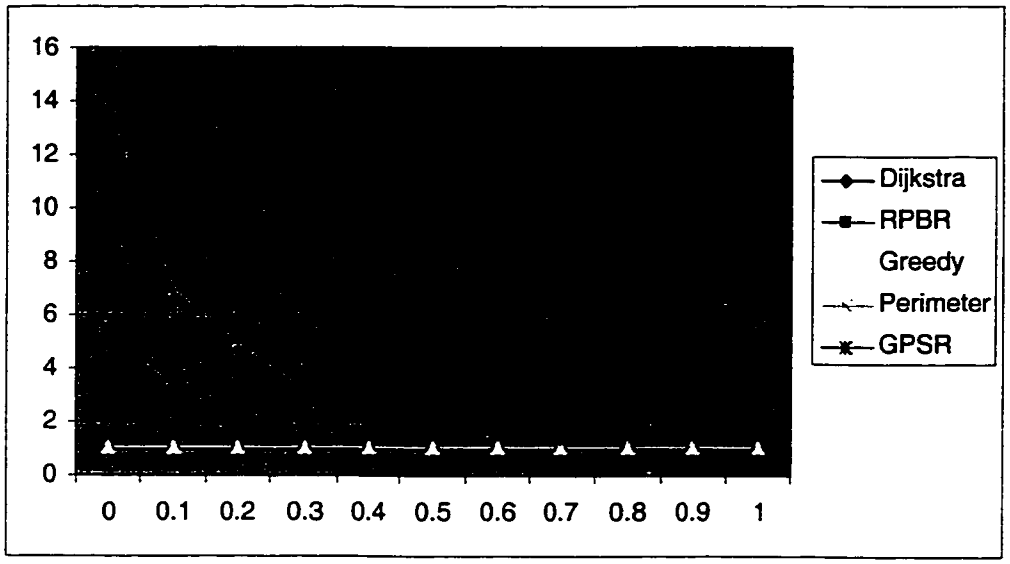


Fig 4.14 Effect of $P_{neighbor}$ on the length of routes.

Figure 4.14 shows the effect of $P_{neighbor}$ on the length of routes produced by the algorithms. As mentioned above, $P_{neighbor}$ being 0 or 1 means that the transmission range is regular. Generally, the length of routes is smaller when $P_{neighbor}$ is larger because nodes can see more neighbors.

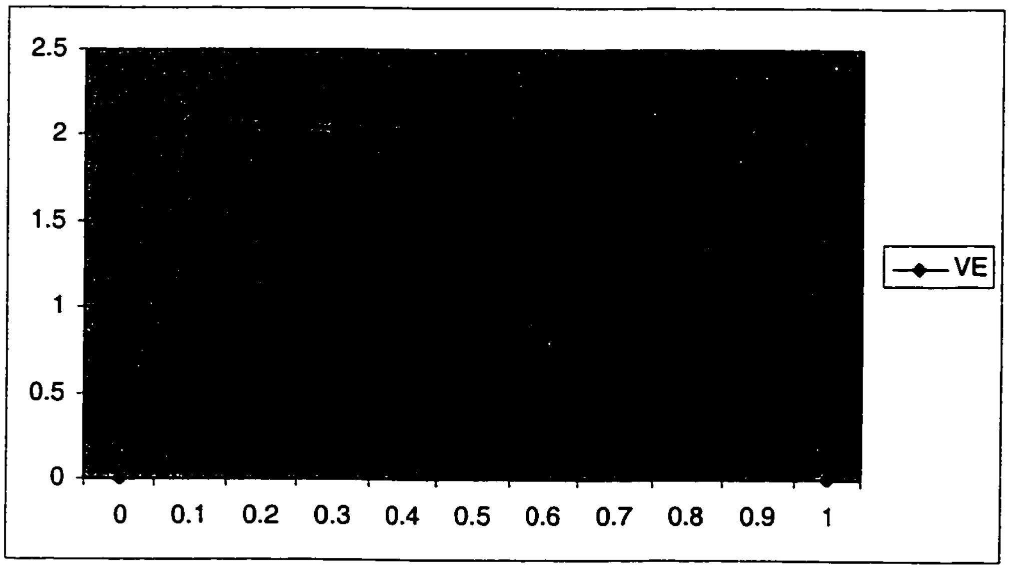
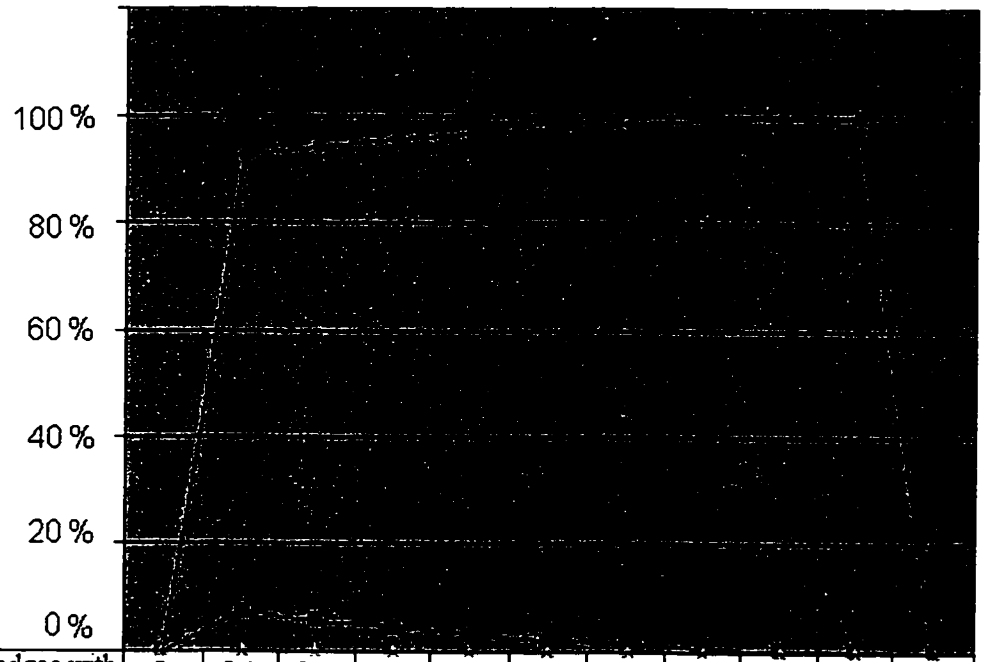


Fig 4.15 Effect of $P_{neighbor}$ on the average length of virtual edges.

Figure 4.15 shows the effect of $P_{neighbor}$ on the average length of virtual edges. When $P_{neighbor}$ equals 0 or 1, there is no virtual edge. The average length of virtual edges is longer when $P_{neighbor}$ is smaller. When $P_{neighbor}$ is small, a node can see fewer nodes and therefore, long virtual edges are likelier to be formed.



% of edges with length	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
—◆— 2	0.00	92.47	94.18	95.42	96.74	97.67	98.43	99.00	99.41	99.68	0.00
—■— 3	0.00	7.35	5.70	4.48	3.21	2.29	1.55	0.99	0.59	0.32	0.00
4	0.00	0.18	0.12	0.10	0.05	0.04	0.02	0.01	0.00	0.00	0.00
—✕— 5	0.000	0.005	0.000	0.001	0.001	0.000	0.000	0.000	0.000	0.000	0.000
Max Length	0	5	4	5	5	4	4	4	4	3	0

Fig 4.16 Effect of $P_{neighbor}$ on the distribution of lengths of virtual edges.

Figure 4.16 shows the effect of $P_{neighbor}$ on the distribution of lengths of virtual edges and the maximum length. At least 92% of virtual edges have length 2 regardless of the value of $P_{neighbor}$. As mentioned before, when $P_{neighbor}$ equals 0 or 1, there is no virtual edge. There are more longer virtual edges when $P_{neighbor}$ is smaller. Along with the increase in $P_{neighbor}$, the percentage of virtual edges with length 3, 4 and 5 decreases and the percentage of virtual edges with length 2 increases. Even though the percentage of virtual edges of length higher than 2 is as high as 8% when $P_{neighbor}$ is 0.1, it is interesting to observe that the overwhelming majority of these long virtual edges have length 3; even in this situation, it is extremely rare to have very long virtual edges.

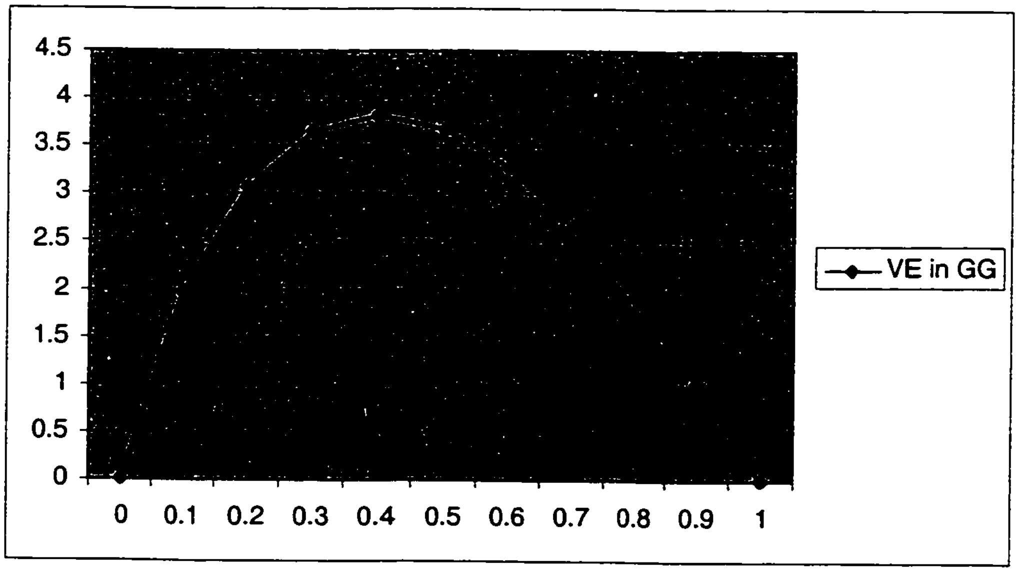


Fig 4.17 Effect of $P_{neighbor}$ on the percentage of virtual edges in GG.

Figure 4.17 shows the effect of $P_{neighbor}$ on the percentage of virtual edges in the Gabriel graph. The percentage of virtual edges in the Gabriel graph is highest when $P_{neighbor}$ is 0.4. Since when $P_{neighbor}$ is 0 or 1, there are no virtual edges, the percentage of virtual edges in the Gabriel graph is 0 in these situations. Also when $P_{neighbor}$ is close to 1, there are unlikely to be many virtual edges in the graph because most nodes within distance R can be seen directly by the node. However, when $P_{neighbor}$ is close to 0, we would expect more and longer virtual edges, so it is curious that the percentage of virtual edges in the Gabriel graph is low.

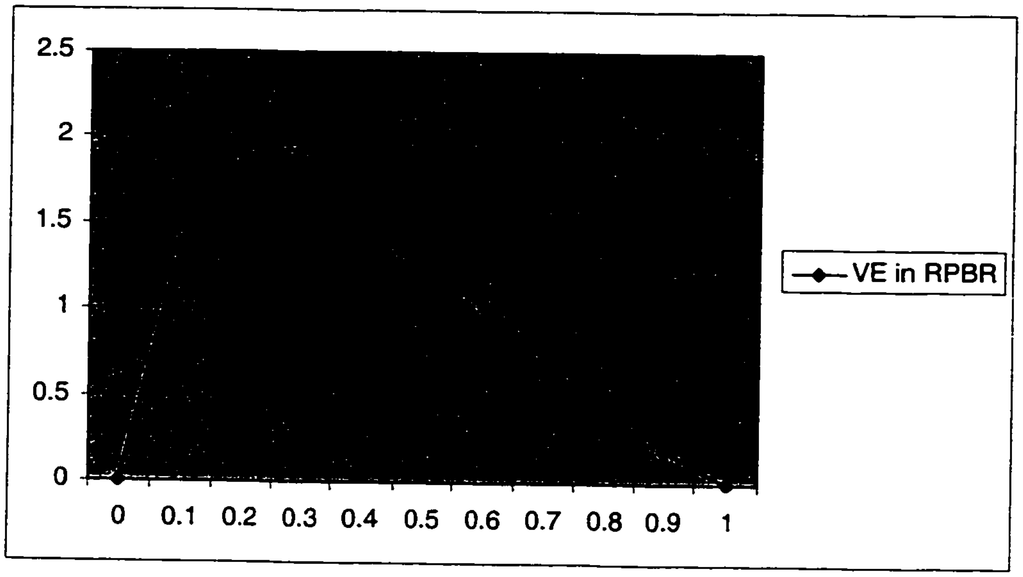


Fig 4.18 Effect of $P_{neighbor}$ on the percentage of virtual edges in RPBR routes.

Figure 4.18 shows the effect of $P_{neighbor}$ on the percentage of virtual edges in the route produced by RPBR. The percentage of virtual edges in the route found by RPBR is highest when $P_{neighbor}$ is 0.3. It follows the pattern of the percentage of virtual edges in the Gabriel graph as expected.

4.2 Analysis summary

We summarize the results of the simulations below:

1.From the point of view of performance, Dijkstra's algorithm always produces the shortest path; Greedy is the second; RPBR and GPSR are the third; perimeter routing produces the longest routes. The routes generated by RPBR are on average a little longer than those generated by GPSR. This is because RPBR uses virtual edges, which have average length two. Since virtual edges only comprise a very small percentage of the RPBR routes, they only increase the length of the routes by a small amount.

2.In all the simulations, Dijkstra's algorithm and RPBR never fail. Greedy routing has the highest failure rate; perimeter routing has the second-highest failure rate, followed by GPSR. In Chapter 3, we explained why Greedy routing is more likely to fail. The reason perimeter routing's failure rate is higher than that of GPSR is because GPSR can switch between the two algorithms, and it fails only when both algorithms fail.

3. The average length of virtual edges is a little bit higher than two; this means that most of the virtual edges have length two. In fact, the percentage of virtual edges with length greater than 2 is less than 8% regardless of density, the *R/r* ratio and the value of $P_{neighbor}$. In all simulations, no virtual edge longer than 6 was ever encountered. When $P_{neighbor}$(the probability of being a neighbor when a node is at distance between r and R) is 0 or 1, there are no virtual edges. The percentage of

virtual edges in the Gabriel graph is highest when $P_{neighbor}$ is 0.4 or when the *R/r* ratio is 1.4.

4.Overall, the percentage of virtual edges in the routes found by RPBR is very low (less than 2%). Furthermore, as mentioned above, most virtual edges correspond to actual paths of length 2. Clearly, the use of virtual edges will only increase the length of the route by a small amount. At the same time, we have the advantage of guaranteeing packet delivery in an irregular transmission range environment.

5. Conclusions

Routing in ad hoc wireless networks must take into account the infrastructureless nature of ad hoc networks as well as the low bandwidth of wireless networks. Position-based routing algorithms attempt to avoid the high cost of flooding to find routes. However, they assume uniform transmission ranges and may fail in the presence of irregular transmission ranges. This report examines the performance of four different routing algorithms for ad hoc networks. We designed and implemented a simulator for generating ad hoc networks with irregular transmission ranges and to run routing algorithms. We ran extensive simulations of Dijkstra's shortest path, Greedy routing, perimeter routing, GPSR and RPBR in order to compare their performance.

RPBR is a position-based routing protocol: it avoids flooding route discovery packets over the network and does not need to maintain routing tables. Also, it can deal with irregular transmission ranges. Both perimeter routing and GPSR may fail with irregular transmission ranges. RPBR can always extract a planar and connected subgraph to perform routing on and thus guarantees the packet delivery.

The RPBR algorithm never fails. It introduces the concept of virtual edges to guarantee the delivery in ad hoc networks with irregular transmission range. It also includes the advantages of the greedy algorithm so its performance is almost as good as GPSR. Since the average length of virtual edges is two, the use of virtual edges can increase the length of routes found by RPBR. However the average percentage of virtual edges in RPBR routes is less than 2%, that means they only increase the length a little.

6. References

[1] Lali Barriere, Pierre Fraigniaud, Lata Narayanan and Jaroslav Opatrny, Robust Position-Based Routing in Wireless Ad-hoc Networks with Irregular Transmission Ranges, 2002.

[2]S. Basagni, I. Chlamtac, V.R.Syrotiuk, and B.A.Woodward, A distance routing effect algorithm for mobility (DREAM). In 4th ACM/IEEE Conference on Mobile Computing and Networking (Mobicom'98), pages 76-84, 1998.

[3] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, Jorge Urrutia, Routing with Guaranteed Delivery in Ad-hoc Wireless Networks. In 3rd Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIALM'99), pages 48-55, 1999.

[4] Edsger W. Dijkstra, A note on two problems in connexion with graphs. Numerische Mathematik, Volume 1, pages 269–271, 1959.

[5] K. Gabriel and R. Sokal, A new statistical approach to geographic variation analysis. System Zoology 18, pages 259-278, 1969.

[6] Zygmunt J. Haas, and Marc R. Pearlman, The Zone Routing Protocol (ZRP) for Ad Hoc Networks. In IEEE Journal on Selected Areas in Communications, pages 1395-1414, Aug. 1999,

[7]C. Hedrick, Routing Information Protocol. RFC1058, June, 1988.

[8]David B. Johnson, Routing in Ad Hoc Networks of Mobile Hosts. In Workshop on Mobile Computing Systems and Applications, (Santa Cruz, CA, U.S.), 1994.

[9] David B. Johnson and David A. Maltz, DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. In Mobile Computing, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153–181, Kluwer Academic Publishers, 1996.

[10] Brad Karp and H.T. Kung, Greedy Perimeter Stateless Routing for Wireless Networks. In 6th ACM Conference on Mobile Computing and Networking (Mobicom'00), 2000.

[11]E. Kranakis, H. Singh, and J.Urrutia, Compass routing on geometric networks. In Proc. 11th Canadian Conference on Computational Geometry, pages 51-54, Vancouver, August 1999.

[12]Y.-B.Ko and N.H.Vaidya, Location-aided routing(LAR) in mobile ad hoc networks. In Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '98), pages 66–75, 1998.

[13] X. Lin and I. Stojmenovic, GEDIR: Loop-free location based routing in wireless networks. In IASTED Int. Conf. on Parallel and Distributed Computing and Systems (PDC'99), pages 1025-1028, 1999.

[14] Katia Obraczka and Gene Tsudik, Multicast Routing Issues in Ad Hoc Networks. In IEEE International Conference on Universal Personal Communications, 1998.

[15]V. Park, and S. Corson,
Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification; IEEE INFOCOM '97, Kobe, Japan, 1997.

[16] C. Perkins, E. M. Belding-Royer, and S. R. Das, Ad Hoc On Demand Distance Vector (AODV) Routing; Internet-draft, draft-ietf-manet-aodv-13.txt, 2003.

[17] C.Perkins, and P. Bhagwat, Highly-dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In SIGCOMM Symposium on Communications Architectures and Protocols, (London, UK), pages 212-225, Sept. 1994.

[18] I. Stojmenovic, Position-based routing in ad hoc networks. In IEEE Communications Magazine, Vol. 40, No. 7, 128-134, July 2002.