Lebanese American University

A Low Degree Vertex Elimination with High Degree Vertex Selection Heuristic for Strongly Connected Dominating and Absorbent Sets in Wireless Ad-Hoc Networks

by

Christine Hovsep Markarian

A Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

School of Art and Science

August 2011

Lebanese American University
School of Arts & Sciences

Thesis Approval Form

Student Name: Christine Markarian I.D.#: 200401710

Thesis Title: A Low Degree Vertex Elimination with High Degree Vertex Selection Heuristic for Strongly Connected Dominating-Absorbent Sets in Wireless Ad-Hoc Networks.

Program: Computer Science

Division/Dept : Computer Science and Mathematics
School : Arts and Sciences.

Approved by : Signatures Redacted

Faisal N. Abu Khzam, Ph.D. (Advisor)
Associate Professor of Computer Science

Signatures Redacted

Abdel Nasser Kassa, Ph.D.
Associate Professor of Mathematics

Signatures Redacted

Azzam Mourad, Ph.D.
Assistant Professor of Computer Science

Date : 10/08/11

THESIS PROJECT COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY

By signing and submitting this license, I, Christine Markarian, grant the Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute my submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. I agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. I also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. I represent that the submission is my original work, and that I have the right to grant the rights contained in this license. I also represent that my submission does not, to the best of my knowledge, infringe upon anyone's copyright. If the submission contains material for which I do not hold copyright, I represent that I have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, I REPRESENT THAT I HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify my name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to my submission.

Name: Christine Markarian

Signature: 

Date: August 2011

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Christine Markarian

Signature: [redacted]

Date: August 2011

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my advisor, Dr. Faisal Abu-Khzam, for his patient guidance, supportive advices, and encouragement throughout the entire work of my thesis. Sincere thanks are also due to my two graduate committee members, Dr. Azzam Mourad and Dr Abdul-Nasser Kassar for their reviewing and advising efforts. A special thanks to the president of the Lebanese American University, Dr. Joseph Jabbra, along with the entire faculty of the department of computer science and mathematics at LAU.

DEDICATION

This work is dedicated to my family. I am what I am today because of them.

A Low Degree Vertex Elimination with High Degree Vertex Selection Heuristic for Strongly Connected Dominating and Absorbent Sets in Wireless Ad-Hoc Networks

Christine Hovsep Markarian

Abstract

In the Strongly Connected Dominating-Absorbent Set problem (SCDAS), we are given a directed graph and asked to find a subset *D* of vertices such that the subgraph induced by *D* is strongly connected and every vertex not in *D* has both an in-neighbor and an out-neighbor in *D*. SCDAS received attention recently because "small" strongly connected dominating-absorbent sets serve as "efficient" virtual backbones in asymmetric wireless networks.

This thesis studies the Minimum SCDAS problem, which seeks a smallest SCDAS in a given digraph. We introduce a new heuristic approach based on a hybrid of low-degree vertex elimination and high-degree vertex selection. Experimental results show that our approach outperforms all previously known algorithms for the SCDAS problem.

Keywords: Disk graph, Dominating set, Absorbent set, Strongly connected, Wireless ad-hoc network, Heuristic, Virtual backbone

TABLE OF CONTENTS

Contents

INTRODUCTION 1
1.1 Overview 1
1.2 Related Work 3
1.3 Thesis Outline 4
THEORETIC BACKGROUND AND EXISTING APPROACHES 5
2.1 Theoretic Background and Terminology 5
2.2 Existing Approaches 7
2.2.1 Dominating-Absorbent Spanning Trees (DAST) 7
2.2.2 Greedy Strongly Connected Component Merging (G-CMA) 7
2.2.3 Exact Algorithm 8
A LOW DEGREE VERTEX ELIMINATION WITH HIGH DEGREE VERTEX SELECTION HEURISTIC (LDHD) 9
3.1 LDHD Description 10
3.2 LDHD Correctness 12
3.3 LDHD Complexity 12
EXPERIMENTAL RESULTS 13
4.1 Random Graph Generators 13
4.1.1 Random Generator I 13
4.1.2 Random Generator II 14
4.1.3 Random Generator III 15
4.2 Simulations 15
4.2.1 Simulation I 15
4.2.2 Simulation II 16
4.2.3 Simulation III 18
REFERENCES 26
Appendix A 29
Appendix B 37

LIST OF TABLES

Table 1: Varying number of nodes and density of a graph 18

Table 2 Summary of experimental results compared to the optimal solution 23

LIST OF FIFURES

Figure 1 A directed graph (DDG) modeling a network..... 6

Figure 2: Impact of Number of Nodes 16

Figure 3: Network Density: Different number of nodes 19

Figure 4: Network Density: Different area size..... 21

Figure 5: Different Transmission Ratios, N=50 22

Figure 6: Different Transmission Ratios, N=100 23

CHAPTER ONE
INTRODUCTION

1.1 Overview

An ad-hoc wireless network is a complex system made up of mobile hosts joined by links for communication i.e. routing. Such a system functions without any fixed infrastructure (access points or base station). An ad-hoc wireless network has been historically used in military applications in which wireless mobile communication systems are used for coordination in battlefields rather than centralized systems which are prone to failure. Later, as Bluetooth and Wireless Internet technologies appeared, ad-hoc wireless networks received attention in applications such as disaster recovery, business, environment monitoring, conferencing, etc [1]

Each mobile host in an ad-hoc wireless network has a *transmission range* specifying the hosts it can communicate with. A host may move to a new location at any time and at any speed which makes the topology of such a network a *dynamic* one. On the other hand, when one mobile host communicates with another mobile host, all other hosts in its transmission range can hear the conversation. Two nodes may not communicate directly if they don't lie within the transmission ranges of each other. Other intermediate hosts are needed to relay the transmission. This is referred to as the *multi hop* characteristic, which must be handled to allow hosts located away from each other to communicate. Moreover, unlike wired networks, ad-hoc wireless networks have limited resources (Ex: mobile hosts are usually battery-powered). This makes wireless links carry *less bandwidth* than wired links. The *dynamic*, *multi hop*, and *limited*

bandwidth characteristics of wireless ad-hoc networks thus make routing a challenging problem.

Routing in wireless ad-hoc networks can be of three types: reactive, proactive or a combination of both. In proactive routing, each node computes and stores all routing information, thus when a route is requested, it can be directly provided. Unlike proactive routing, reactive routing asks a node to compute routing information for a specific destination only when needed, i.e. *on demand* routing. The third type of routing asks only some hosts to keep some partial information. *Proactive* or *reactive* routing decisions are thus made.

Lately, an approach based on *virtual backbone* replaced existing types and proved to be a promising routing approach. A virtual-backbone consists of selected hosts that keep routing information and are responsible for all transmissions in the network. This virtual backbone facilitates the routing process since messages need not be broadcast to all the nodes in the network. Only nodes in the backbone become responsible for routing messages. Clearly, this reduces routing overhead.

A wireless ad-hoc network can be modeled by a graph G = (*V, E*), where the elements of *V* represent the mobile hosts and the elements of *E* represent the communication links. *Dominating-sets* have served as “efficient” virtual backbones for ad-hoc wireless networks. A subset of vertices of a given graph is a *dominating* set if every vertex is either in the subset or has some neighbor in it. A *small* and *connected* dominating set is clearly needed to facilitate the routing process within the vertices in the set.

The problem of finding a minimum connected dominating set has been extensively studied in unit disc graphs (UDG), in which all nodes in the network have the same transmission

range. Such graphs are undirected graphs. In practice, however, the transmission ranges of all nodes in the network need not be the same due to differences in power and functionality. Therefore, a directed disc graph (DDG) would best model such a network.

In this thesis, we model a wireless ad-hoc network as a directed disc graph and seek a strongly connected dominating-absorbent set (SCDAS) rather than a connected dominating set in an undirected graph. Unfortunately, the strongly connected dominating-absorbent set problem (SCDAS) is NP-hard since connected dominating set problem (CDS) is NP-hard and CDS is a special case of SCDAS. [18]

We propose a new heuristic approach for the Minimum SCDAS problem, which seeks a smallest SCDAS in a directed graph. We refer to it as "low degree vertex elimination with high-degree vertex selection heuristic (LDHD)". Experimental results show that LDHD outperforms all previously known algorithms for the SCDAS problem.

1.2 Related Work

The construction of virtual backbone problem based on *dominating sets* in wireless ad-hoc networks has been extensively studied in unit disc graphs (UDGs) [4] to [17]. Only recently, the problem has been explored in directed disc graphs. The authors in [18] and [19] extend their previous work on UDGs called the marking process, to asymmetric networks where nodes have different transmission ranges. The main concept of their process is that every time a node discovers that it has two neighbors that are not joined by a directed edge, it becomes part of the solution, i.e. in the strongly connected dominating absorbent set. (SCDAS) They later added more rules to decrease the number of vertices in the constructed SCDAS. No approximation ratio has been given, however. Lately, the authors in [1] and [3] proposed constant

approximation algorithms for SCDAS. The authors in [3] apply Breadth First Search and Steiner Nodes techniques. The approximation algorithm proposed in [1] works only when the transmission range ratio is bounded by some constant. Later, a polynomial-time, 3 log n approximation algorithm for the same algorithm of [1] is proposed in [21]. The authors in [1] also propose two heuristics for SCDAS. The two heuristics first find a Dominating Absorbent Set then greedily use additional nodes to make it strongly connected.

1.3 Thesis Outline

The remaining of this thesis is structured as follows: Theoretic background and algorithms for existing approaches for SCDAS are presented in chapter 2. The Low Degree Vertex Elimination with High Degree Vertex Selection heuristic, (LDHD), is proposed in chapter 3. In chapter 4, we discuss the experimental study conducted to evaluate our approach along with the experimental results. We conclude with some direction for future work in Chapter 5.

CHAPTER TWO
THEORETIC BACKGROUND AND EXISTING APPROACHES

2.1 Theoretic Background and Terminology

In this thesis, a *graph* is a set of *nodes/hosts* or *vertices* joined by links called *edges*. The terms *vertices*, *nodes* and *hosts* are used interchangeably in this thesis. If *e*= (*u*, *v*) is an edge, then vertices *u* and *v* are called *end nodes* of e. The two nodes are called *neighbors* and are *adjacent* to each other. Node *u* and edge *e* are *incident* to each other. If *u* =*v*, then edge *e* is a *loop*. *Parallel edges* are two edges that connect the same end nodes. A *simple graph* is a graph that does not contain any loops or parallel edges.

If two edges share a common node, then they are *adjacent* to each other. A *weighted graph* is a special graph in which a number has been assigned to each edge. The number of edges incident to a node is its *degree*. A *complete graph* is a graph in which an edge exists between any two distinct vertices. The edge is referred to as a *directed edge* if its end nodes are an ordered pair. An *incoming edge* to v*, e*= (*u*, *v*) is an edge directed from *u* to *v* where *u* is an *in-neighbor* of *v*, and *v* is an *out-neighbor* of *u*. If *u* is an in-neighbor of *v* then we say *u dominates v* and *v absorbs u.* The *in-degree* of a node *u* is the cardinality of the set of *u*'s in-neighbors and the *out-degree* of a node *u* is the cardinality of the set of *u*'s out-neighbors. In a *directed graph*, or *digraph*, all edges are directed. In an *undirected graph*, edges are not directed.

Fig. 1 shows a *simple directed* graph modeling an *asymmetric* network, a network where nodes have different transmission ranges. Each node's transmission range is represented as a dotted circle and the directions on the edges show the difference between *bidirectional* and

unidirectional edges. [1]. For an edge *e*= (*u*, *v*), *e* is said to be *unidirectional* if there is a directed edge from *u* to *v* but no directed edge from *v* to *u*. *e* is said to be *bidirectional* if there is a directed edge from *u* to *v* and another from *v* to *u*.

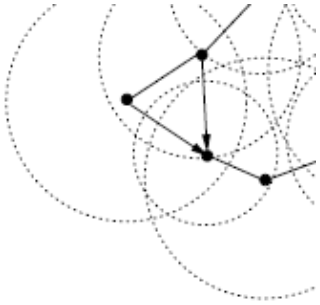

Figure 1 A directed graph (DDG) modeling a network

A *walk* is a sequence of nodes in which two consecutive nodes are the end nodes of an edge in the graph. A *path* is a walk in which each node is distinct. A *directed path* is a path in a digraph. If there is a path between each pair of nodes in a graph, the graph is said to be *connected*, otherwise the graph is *disconnected*. A directed graph in which every two nodes are joined by a directed path is a *strongly connected* graph. A *subgraph* of a graph is a subset of the nodes and the edges of the graph.

A maximal connected subgraph of G is a connected *component* of G. A subgraph having the entire nodes of G is called a *spanning subgraph* of G.

A *tree* is a connected graph which has no cycles. A graph without any cycle is a *forest*. A tree that is a spanning subgraph of a graph G is a *spanning tree* of G. A forest that is a spanning subgraph of a graph G is a *spanning forest* of G. [22]

2.2 Existing Approaches

Currently, the best known approaches for the SCDAS problem are: (1) the Dominating-Absorbent Spanning Trees (DAST), (2) Greedy Strongly Connected Component Merging algorithm (G-CMA), both proposed in [1], and (3) the only Exact algorithm for the SCDAS problem, the brute force algorithm of complexity 2^n, which enumerates all strongly connected dominating-absorbent sub graphs and selects the minimum among them.

2.2.1 Dominating-Absorbent Spanning Trees (DAST)

The Dominating-Absorbent Spanning Trees (DAST) algorithm constructs two spanning trees rooted at some node *r,* one outgoing and another incoming, then takes the union of the two trees, excluding the leaves, as a *strongly connected dominating absorbent set*.

The algorithm first forms a dominating set (DS) by constructing a spanning tree rooted at some node *r*. It uses a simple coloring technique which colors a vertex black once it is chosen to be in the DS. Then it colors all its out neighbors gray. The algorithm then builds an absorbent set (AS) by reversing the edges and constructing another spanning tree rooted at the same node *r* using the same coloring technique. The algorithm terminates by taking the union of the two trees excluding the leaves.

2.2.2 Greedy Strongly Connected Component Merging (G-CMA)

The Greedy Strongly Connected Component Merging algorithm (G-CMA) constructs a *strongly connected dominating set* (SCDAS) in two stages. First, it finds a dominating absorbent set then uses additional nodes to make the set strongly connected. To make the set strongly connected, G-CMA merges repeatedly two pair of strongly connected components via shortest path between them until one strongly connected component is left.

2.2.3 Exact Algorithm

There is no exact algorithm for the SCDAS problem in the literature. Thus, the 2^n brute force algorithm which enumerates all sub graphs and selects the minimum among all strongly connected dominating-absorbent sub graphs is the only way to find an exact solution to the SCDAS problem.

Since the brute force algorithm is very slow especially when the number of vertices is large, we try to make it as fast as we can by generating the sub graphs from the least order sub graph to the greatest one. The algorithm thus stops as soon as it finds the first sub graph which is strongly connected dominating and absorbent.

The algorithm we use to generate the sub graphs in increasing order is an algorithm known as the Banker's Sequence algorithm presented by Loughry in [20] which is more efficient than the other two existing algorithms for generating all sub graphs of a given graph namely the Lexicographic Ordering algorithm and Gray Codes algorithm.

CHAPTER THREE

A LOW DEGREE VERTEX ELIMINATION WITH HIGH DEGREE VERTEX SELECTION HEURISTIC (LDHD)

In this chapter, we propose a new heuristic approach that is a hybrid of low-degree vertex elimination and high-degree vertex selection. We refer to this approach as LDHD.

A *low-degree* vertex is more likely to *dominate* and *absorb* fewer vertices than vertices of higher degree. On the other hand, a *high-degree* vertex with relatively more *in* and *out neighbors* can absorb and dominate more vertices than other vertices. Thus, a "good" solution will most likely contain *many* high-degree vertices and *few* low-degree vertices.

However, since the *solution set* must not only dominate and absorb all vertices in the graph, but also "*be strongly connected*, " the above statement would not be enough. In other words, despite their disability to absorb and dominate relatively *many* vertices, low-degree vertices might be used as *intermediary* vertices to strongly connect the vertices in an "optimal" solution set. For this reason, LDHD makes use of the characteristics of both: "low degree" vertices and "high degree" vertices while taking into consideration the contribution of low-degree vertices in providing the required "*strong connectivity property*".

Rather than discovering the *low-degree* vertices that contribute in the "*strong connectivity property*", the algorithm starts by deleting vertices that Do Not contribute to strong connectivity property. This can easily be done by removing a *low-degree* vertex from the graph and checking if the remaining graph forms a strongly connected graph. If it does, then the vertex can be deleted from the graph. Moreover, when such a vertex is deleted, it has to be

dominated and absorbed by a subset of the remaining vertices. Therefore, *high-degree in* and *out-neighbors* are selected to absorb and dominate this vertex thus making use of the *characteristic of high-degree vertices.*

The main idea of LDHD can thus be summarized as follows: A low degree vertex can be deleted from the graph if its removal does not *strongly* disconnect the remaining graph. Otherwise, it must belong to the *strongly connected dominating-absorbent set* (SCDAS). Once such a vertex is deleted, high degree vertices are selected to *dominate* and *absorb* this vertex. Once a vertex is selected, it can never be deleted.

3.1 LDHD Description

Given a *strongly connected directed graph*, G (V, E), LDHD finds a *strongly connected dominating-absorbent set*, *D, as follows:*

A vertex is either decided to be in the solution and colored **black**, or decided not to be in the solution and colored **red**. A **white** vertex is not yet decided.

Initially, all vertices are in *D,* the solution set, and colored **white**. Some *preprocessing* is done as follows:

- The *in-neighbor* of a vertex with *in-degree* 1 is colored **black.**
- The *out-neighbor* of a vertex with *out-degree* 1 is colored **black.**

We repeatedly select a **white** vertex, *v*, with minimum degree. If G [*D* - {*v*}] is not strongly connected, we color *v* **black** and do *preprocessing*. Otherwise, we remove *v* from *D*, color it **red**, and update the degrees of its neighbors.

Then, if none of *v*'s *in-neighbors* is **black,** we select the *in-neighbor* with maximum degree and color it **black**. Similarly, if none of *v*'s *out-neighbors* is **black,** we select the *out - neighbor* with maximum degree and color it **black**. At this step, *preprocessing* is done as well.

The algorithm ends when no **white** vertices remain. All vertices now are either **black**, i.e. in the solution, or **red**, i.e. not in the solution but are absorbed and dominated by **black** vertices. Thus, the Set *D* of **black** vertices forms a *Strongly Connected Dominating-Absorbent Set.*

A LOW DEGREE VERTEX ELIMINATION WITH HIGH DEGREE VERTEX SELECTION HEURISTIC (LDHD)

Input: A Strongly Connected Directed Graph, G (*V*, *E*),
Output: *A* Strongly Connected Dominating Absorbent Set, *D*

```
D ← V
All vertices are initially colored white
Do preprocessing
While D has white vertices
{
   Select a white vertex v with minimum degree
   If G [ D - {v}] is strongly disconnected
   {
       Color v black
       Do preprocessing
   }
   Else
   {
       Color v red // v is not in the solution set, D
       Update the degrees of its neighbors (decrement by 1)
       If none of v's in - neighbors is black
           Select the in - neighbor with maximum degree and color it black.
       If none of v's out - neighbors is black
           Select the out - neighbor with maximum degree and color it black.
       Do preprocessing
    }
}
```

3.2 LDHD Correctness

To show that the set *D* forms a SCDAS, we use induction as follows.

At step 0: *D* =*V* is clearly a SCDAS since the input graph is *strongly connected*.

At step i: Let *D* be a *strongly connected dominating-absorbent set*.

At step i + 1: We remove from *D* (color red) a vertex *v* only if removing it doesn't *strongly disconnect* the graph.

Moreover, to remove *v*, the algorithm makes sure that *v* is *dominated* i.e. has a black *in-neighbor*. If not, it selects the highest degree in-neighbor and colors it black. Similarly, the algorithm makes sure that *v* is *absorbed* i.e. has a black *out-neighbor*. If not, it selects the highest degree out-neighbor and colors it black.

Thus every vertex not in *D* will be both *dominated* and *absorbed* and the set *D* will be strongly connected. This implies that S is a *strongly connected dominating-absorbent set*.

3.3 LDHD Complexity

The while loop in the algorithm is executed at most *n* - 1 times because at each step we are either eliminating or keeping at least one node. To check if removing a vertex *v* disconnects the graph requires running either Depth First Search or Breadth First Search twice. This takes 2 * O (*m* + *n*) time where *n* is the number of vertices and *m* is the number of edges. Thus, LDHD would have an overall complexity of O (*mn*).

CHAPTER FOUR
EXPERIMENTAL RESULTS

To have an efficient evaluation of our proposed scheme, we implemented all three heuristics, LDHD, DAST, and G-CMA in the same programming language, C, and used the same data structures (adjacency lists were used to represent the graphs). Running times were measured on an Intel Core 2 duo CPU of 2.0 GHz with 4 GB memory.

Three simulations were conducted using three random generators. The first section discusses the random graph generators, and the second section presents the simulations and experimental results.

4.1 Random Graph Generators

To measure the performance of LDHD and show its efficiency on almost all types of graphs, we present three random generators that were implemented and used to generate input graphs for the algorithms we implemented. The third generator is a general random generator used by most previous work on the problem. The first two are proposed to guarantee that LDHD outperforms all existing approaches no matter what the input graph is. Note that the generated input graph must be *a simple directed strongly connected graph*

4.1.1 Random Generator I

The algorithm first selects an arbitrary node u where $0 < u < N-1$. A random number x, where $1 < x < N-1$ is chosen. It then chooses randomly x nodes, $s_1,s_2,s_3,s_4,.....s_x$ and adds an edge $(s, s_i) \forall s_i \in \{ s_1,s_2,s_3,s_4,.....s_x\}$. Then the algorithm selects any node $s_i \in \{ s_1,s_2,s_3,s_4,.....s_x\}$ and a random number y, where $1 < y < N-1-x$. It randomly chooses y nodes, $v_1,v_2,v_3,v_4,.....v_y$ and adds an edge $(u_i, v_i) \forall v_i \in \{ v_1,v_2,v_3,v_4,.....v_x\}$. The algorithm continues as such until a directed tree

rooted at u spanning all vertices is formed. To make the graph strongly connected, a directed tree rooted at u but with reverse edges is then formed in a similar way. The resulting union of the two trees forms a Random Strongly Connected Directed Graph.

4.1.2 Random Generator II

Although the above algorithm generates a Random Strongly Connected Directed Graph, another version of the RANDOM GENERATOR was implemented and used as a basis for our testing to guarantee the efficiency of our proposed algorithm no matter what the input graph is. In other words, RANDOM GENERATOR II attempts to generate the most Random Strongly Connected Graphs which are not characterized by any specific structure like those of RANDOM GENERATOR I.

The main idea of RANDOM GENERATOR II is that it starts with a complete directed graph and removes edges randomly to generate a user specified dense/sparse graphs. In other words, we decide and select a number of R edges to be removed where $0 < R < E-N-1$ and the algorithm generates a Random Strongly Connected Graph with N vertices and E - R edges, where $E = N (N-1)$. Obviously, this was needed to be able to compare our results on different graphs each with different average degree, i.e. different densities.

Since the above algorithm removes an edge and checks whether the remaining graph is strongly connected, its performance was very poor when implemented specially in the case of generating sparse graphs. Thus, an enhancement was made to improve its performance. Instead of removing an edge at a time and checking whether the remaining graph is strongly connected, N/10 edges were selected at a time (Note: Experiments showed that 10 is more appropriate than some other number such as 5, 6, 7, 8, 9..), if removing these edges leaves the

graph strongly connected; we just remove the selected N/10 edges, otherwise we randomly select another set of N/10 edges and do the same. The algorithm stops either when R edges were removed or when successively 1000 random sets of N/10 edges were removed and the remaining graph was strongly disconnected.

4.1.3 Random Generator III

To generate a random directed graph or an asymmetric network, *n* nodes with distinct identity numbers between 1 and *n* are located in a limited square area in the Euclidean plane randomly. Each node chooses a random transmission range which is bounded by some maximum and minimum values for transmission ranges. A directed edge is added from node *u* to node *v* if the Euclidean distance between *u* and *v* is less than the transmission range of *u*. If the generated network is strongly connected, we use it as one instance, otherwise we discard it.

4.2 Simulations

The three random graph generators presented above are used to conduct three simulations respectively. The first two simulations are based on general directed graphs, whereas the third simulation is based on unit disc graphs representing wireless ad-hoc networks.

4.2.1 Simulation I

Random Generator I is used to generate input graphs of *n* vertices where *n* changes from 100 to 1000 with an increase of 100. For each value of *n*, we study 100 instances and take the average of their results. **Fig 2** shows the differences in the size of the Strongly Connected Dominating-Absorbent Sets (SCDAS) constructed by each of the three heuristics, G-CMA, DAST, and LDHD as the number of vertices increases.

As shown in **fig 2,** LDHD constructs the smallest SCDAS for all values of *n*. Moreover, in average, the size of the SCDAS constructed by DAST is double that of LDHD and the SCDAS constructed by G-CMA is 1.3 times that of LDHD.

This means that LDHD is better than both G-CMA and DAST for all values of *n*.

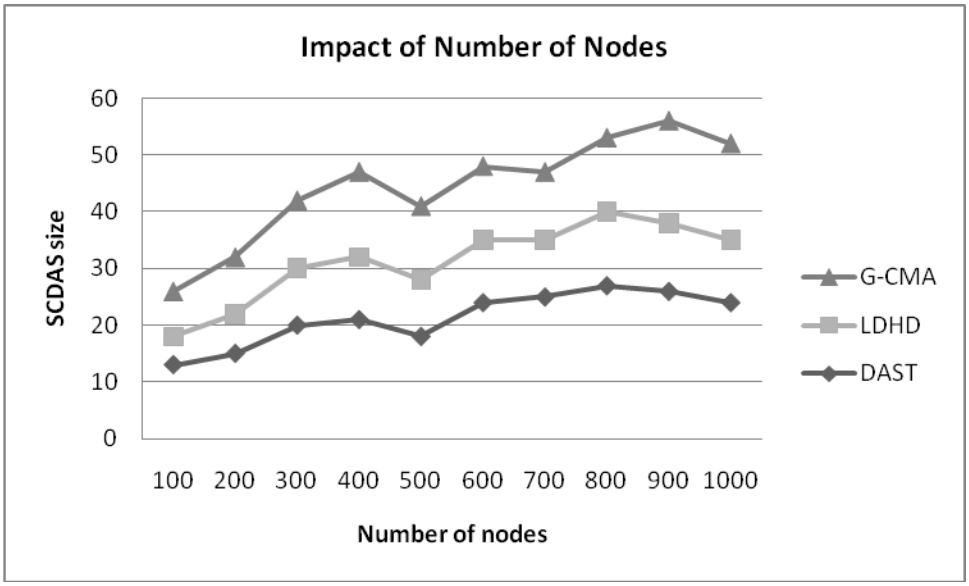

Figure 2: Impact of Number of Nodes

4.2.2 Simulation II

Unlike *Random Generator I*, *Random Generator II* generates more random graphs which are not characterized by any specific structure and could be generated with specific parameters. To properly compare the size of the SCDAS constructed by each of the heuristics and the time (in seconds) taken to construct the set, we specify two parameters for the generator: (1) Number of Nodes (2) Average Degree (or how sparse/dense the graphs is). We vary the number of vertices, *n*, between 200 and 1400 as we increase by 200 and for each *n*, we specify 3 to 4 different Average Degrees which range from 20% of *n* to 80% of *n*. For each *n* and each Average degree, we investigate 100 instances and take the average of the results of each.

As shown in **Table 1**, in average, DAST outperforms both G-CMA and LDHD in terms of the time taken to construct a SCDAS. While DAST takes an average of 0.048 seconds to construct a SCDAS, LDHD takes an average of 6.624 seconds and G-CMA takes an average of 26.005 seconds. However, the difference in the size of SCDAS constructed in each of the three approaches is apparent as DAST constructs the largest SCDAS among the three for all values of *n* and all values of Average Degrees. On the other hand, LDHD outperforms both approaches and constructs a SCDAS of size 2.3 times that constructed by DAST with a difference of only 6 seconds at most in the time taken to construct a SCDAS in average. As for the difference in performance between G-CMA and DAST, GCMA performs poorly both in terms of the time taken to construct a SCDAS and in terms of the size of a SCDAS. As **table 1** show, G-CMA is about 4 times slower than LDHD and constructs a SCDAS of size about 1.24 times that of G-CMA in average.

As a summary, we can say that both G-CMA and DAST either perform poorly in terms of the size of the SCDAS constructed or the time taken to construct such set. Although, DAST can be faster than LDHD, it is more than 2 times worse than LDHD in terms of the size of the SCDAS. As for G-CMA, LDHD outperforms it both in terms of computational time and the size of the SCDAS constructed.

Number of Nodes	***Average Degree***	**DAST (s)**	**DAST (SCDAS size)**	**G-CMA (s)**	**G-CMA (SCDAS size)**	**LDHD (s)**	**LDHD (SCDAS size)**
200	***159***	0.015	8	0.281	4	0.109	4
200	***119***	0.015	11	0.718	8	0.078	8

200	***79***	0.000	21	1.747	14	0.047	9
200	***39***	0.000	46	3.651	23	0.031	16
400	***319***	0.016	11	1.622	5	0.827	4
400	***239***	0.015	16	3.463	9	0.578	7
400	***159***	0.016	24	7.379	14	0.390	11
400	***79***	0.016	58	15.850	23	0.203	19
600	***479***	0.031	10	3.790	6	2.714	4
600	***359***	0.016	18	7.379	8	1.934	7
600	***239***	0.015	27	14.165	13	1.232	12
600	***119***	0.015	58	57.283	29	0.608	22
800	***639***	0.062	10	10.796	7	6.458	6
800	***479***	0.047	18	20.030	11	4.587	6
800	***319***	0.031	27	37.362	17	2.886	11
800	***159***	0.032	56	88.624	27	1.451	23
1,000	***799***	0.078	11	17.035	7	12.621	6
1,000	***599***	0.062	16	27.643	10	8.908	7
1,000	***399***	0.062	26	32.308	12	5.600	13
1,200	***949***	0.125	12	15.881	6	20.733	5
1,200	***699***	0.094	21	37.004	9	14.430	8
1,200	***449***	0.062	30	76.116	16	8.783	13
1,400	***1113***	0.156	14	26.926	6	33.212	6
1,400	***827***	0.125	18	59.572	11	22.901	9
1,400	***541***	0.094	30	83.490	14	14.274	13

Table 1: Varying number of nodes and density of a graph

4.2.3 Simulation III

In the third experiment, we measure the performance of each approach under the effect of two network parameters: Network Density and Transmission Ratio.

1. Network Density

We vary network density in two ways:

a. Different numbers of nodes in a fixed area

b. Different area sizes for a fixed number of nodes

2. Transmission ratio, $k=Tr_{max}/Tr_{min}$, where Tr_{max} is the maximum transmission range and Tr_{min} is the minimum transmission range.

Simulation for each performance measure is repeated 100 times for each instance of every network parameter and the average result is taken. Note that since all three heuristics perform almost the same in terms of computational time (at most 0.6 seconds), comparison in what follows is based on the size of the SCDAS constructed by each heuristic

4.2.3.1 Network Density: Different number of nodes

To compare the size of the *strongly connected dominating-absorbent sets* constructed by each of the four approaches, we deploy N vertices in a 1000m × 1000m area. N changes between 10 and 130 with an increase of 10. The nodes select their transmission ranges from the interval [Tr_{min}=200m, Tr_{max}600m], where Tr_{min} is the minimum transmission range and Tr_{max} is the maximum transmission range.

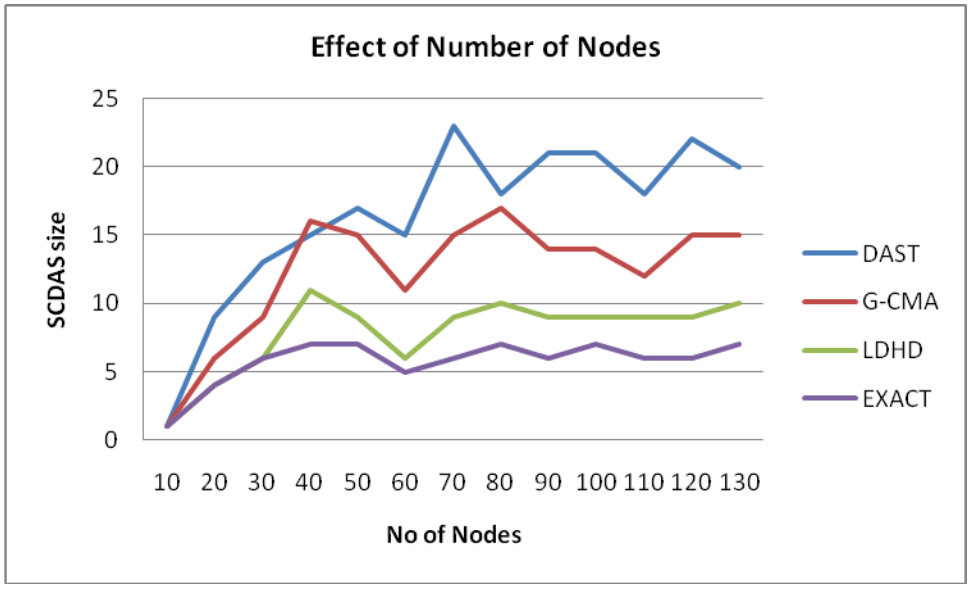

Figure 3: Network Density: Different number of nodes

Clearly, as **fig 3** shows, the SCDAS size becomes larger in all four approaches as the number of nodes increases. This might not be surprising since, when there are few nodes in the

network, the nodes may be far from each other and more nodes are needed to absorb and dominate the nodes in the network.

Fig 3 also shows the difference in the size of the *strongly connected dominating sets* (SCDAS) constructed by each of the four approaches. Obviously, DAST constructs the largest such set for all values of N whereas LDHD constructs a SCDAS of size most optimal compared to that of the EXACT. The size of the SCDAS constructed by LDHD is at most 1.57 times that of the EXACT.

Moreover, compared to the optimal size of the SCDAS constructed by the EXACT algorithm, DAST constructs a SCDAS of size at most 3.8 times that of the EXACT while G-CMA constructs a SCDAS of size at most 2.42 times that of the EXACT.

We can thus notice that LDHD performs, in average, 1.5 times better than G-CMA and 2.08 times better than DAST. Moreover, LDHD is the closest to the optimal solution and almost never exceeds double the size of the optimal solution.

4.2.3.2 Network Density: Different area size

To study the effect of varying the area in the performance of each approach, we deploy a fixed number of vertices, N = 50. The nodes select their transmission ranges from the interval [Tr_{min}=200m, Tr_{max}600m], where Tr_{min} is the minimum transmission range and Tr_{max} is the maximum transmission range. Area varies from 600m x 600m to 1400m x 1400m.

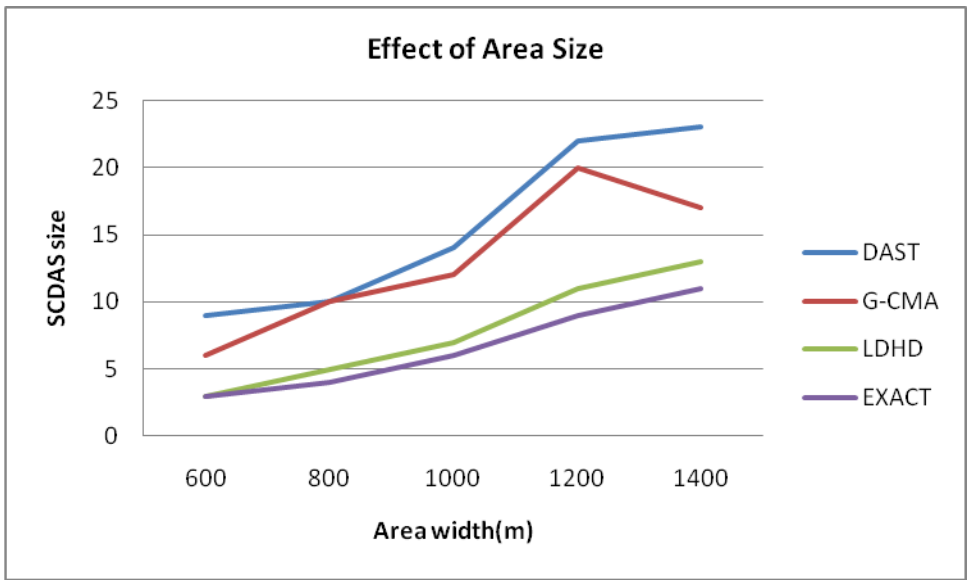

Figure 4: Network Density: Different area size

As shown in **fig 4**, DAST constructs the largest *strongly connected dominating-absorbent set* for all area widths whereas LDHD constructs the smallest such set. While LDHD constructs a *strongly connected dominating-absorbent set* (SCDAS) of size at most 1.25 times that of the EXACT, DAST constructs a SCDAS of size at most 3 times that of the EXACT and G-CMA at most 2.5 times that of the EXACT.

4.2.3.3 Transmission ratio

We also study the effect of varying the transmission ratio, $k=Tr_{max}/Tr_{min}$, where Tr_{max} is the maximum transmission range and Tr_{min} is the minimum transmission range, on the size of the *strongly connected dominating-absorbent sets* (SCDAS) constructed by each of the four approaches. We conduct two experiments. In the first experiment, we randomly locate 50 nodes in a fixed 1000m × 1000m area and vary k as follows: We fix Tr_{max} = 1000m and vary Tr_{min} between 200m and 1000m with an increment of 200 for k=1 to 5.

In the second experiment, we measure the performances on a larger network and randomly locate 100 nodes in a fixed 1200m×1200m area and vary *k* as follows. We fix Tr_{max} = 1200m and vary Tr_{min} between 200m and 1200m with an increment of 200 for k=1 to 6.

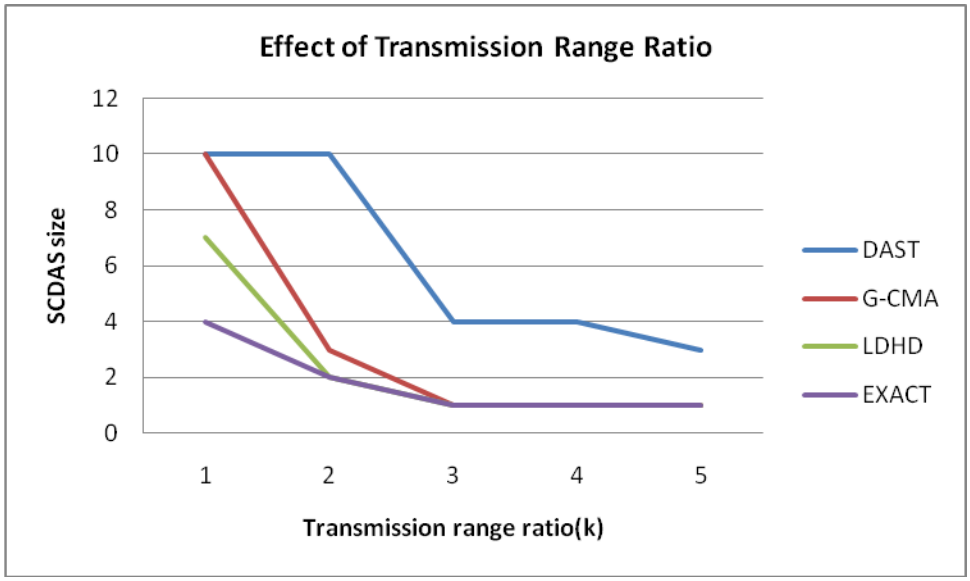

Figure 5: Different Transmission Ratios, N=50

The results of the first experiment are shown in **fig 5.** Obviously as **fig 5** shows, DAST constructs a larger SCDAS than G-CMA does throughout all the interval of *k* whereas LDHD constructs the smallest SCDAS among all three approaches, G-CMA, DAST, LDHD.

DAST constructs a SCDAS of size at most 5 times that of LDHD and G-CMA constructs a SCDAS of size at most 2.5 times that of LDHD. Moreover, LDHD constructs a SCDAS of size at most 1.75 times that of the EXACT.

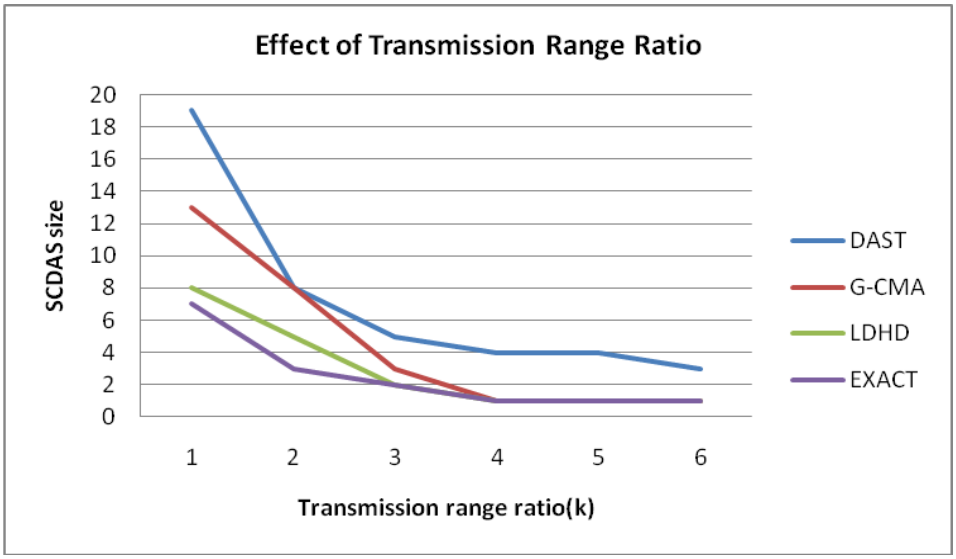

Figure 6: Different Transmission Ratios, N=100

Fig 6 shows the performance results for the second experiment. DAST constructs a SCDAS of size at most 4 times that of LDHD and G-CMA constructs a SCDAS of size at most 2.67 times that of LDHD. Moreover, LDHD constructs a SCDAS of size at most 1.6 times that of the EXACT.

Table 2 shows a summary of the results obtained in all four performance measures. Performance in the worst case of each of DAST, G-CMA, and LDHD is compared to the EXACT solution.

Performance Measure	DAST	G-CMA	LDHD
Number of Nodes	3.8 times larger	2.42 times larger	1.57 times larger
Area Width	3 times larger	2.5 times larger	1.25 times larger
Transmission ratio (Experiment 1)	5 times larger	2.5 times larger	1.75 times larger
Transmission ratio (Experiment 2)	4 times larger	2.67 times larger	1.6 times larger

Table 2 Summary of experimental results compared to the optimal solution

CHAPTER FIVE
CONCLUSION

The problem of constructing a virtual backbone in a wireless ad-hoc network has seen a considerable attention in homogenous networks where all nodes have the same transmission range. Although in practice nodes may not be homogenous, only recently, few researches have studied the problem in heterogeneous networks where nodes have different transmission ranges.

This thesis addressed the problem, Minimum Strongly Connected Dominating-Absorbent Set (SCDAS), in heterogeneous wireless ad-hoc networks and proposed a new heuristic which we refer to as Low Degree Vertex Elimination with High Degree Vertex Selection heuristic (LDHD). LDHD follows a different approach than all existing approaches for the SCDAS problem. Rather than constructing a dominating-absorbent set then strongly connecting it via extra nodes, LDHD provides a sufficient virtual backbone at any time of the algorithm by making sure that, at every step, the constructed set is strongly connected. LDHD is a hybrid of two greedy approaches: low degree vertex elimination and high degree vertex selection. A low degree vertex is deleted if removing it doesn't strongly disconnect the graph. Once a high degree vertex is deleted, high degree neighbors dominate and absorb it.

Despite its simplicity, LDHD proves to perform the best among all existing approaches for the SCDAS problem as investigated in the simulations. Moreover, the results of the simulations show that the SCDAS formed by LDHD is very close to the optimal solution constructed by an EXACT algorithm.

Through simulations, we also showed how LDHD outperforms the best approximation algorithm (3 *ln n* approximation) for the SCDAS problem [21]. Thus, future work might be directed towards giving an approximation bound for the SCDAS constructed by the proposed scheme, LDHD, which would break the 3 *ln n* barrier.

REFERENCES

[1] M. Park, C. Wang, J. Willson, M.T. Thai, W. Wu, and A. Farago, "A dominating and absorbent set in wireless ad-hoc networks with different transmission ranges," in *International Symposium on Mobile Ad Hoc Networking and Computing(Mobihoc),* Canada, 2007, pp. 22 -31.

[2] M.R. Garey and D.S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. [Online]. Available: http://elib.tu-darmstadt.de/tocs/125242654.pdf

[3] M. T. Thai, R. Tiwari, and D. Zhu Du, "On construction of virtual backbone in wireless ad hoc networks with unidirectional links," *IEEE Trans. on Mobile Computing,* vol. 7, (9), pp. 1098 - 1109, 2008.

[4] Y. Li, S. Zhu, M.T. Thai, and D.-Z. Du, "Localized construction of connected dominating set in wireless networks," *National Science Foundation International Workshop on Theoretical Aspects of Wireless Ad Hoc, Sensor and Peer-to-Peer Networks*, Chicago, 2004.

[5] P.J. Wan, K.M. Alzoubi, and O. Frieder, "Distributed construction on connected dominating set in wireless ad hoc networks," in *IEEE International Conference on Computer Communications,* Chicago, 2004, pp. 141-149.

[6] Y. Li, M.T. Thai, F. Wang, C.-W. Yi, P.J. Wang, and D.Z. Du, "On greedy construction of connected dominating sets in wireless networks," *Wireless Communications and Mobile Computing*, vol. 5, (88), pp. 927-932, 2005.

[7] M. Cardei, M.X. Cheng, X. Cheng, and D.-Z. Du, "Connected domination in ad hoc wireless networks," *Computer Science and Informatics (CSI)*, vol. 1, pp. 251-255, 2002.

[8] S. Guha and S. Khuller, “Approximation algorithms for connected dominating sets,” *Algorithmica*, vol. 20, (4), pp. 374-387, 1998

[9] L. Ruan, H. Du, X. Jia, W. Wu, Y. Li, and L.I. Ko, “A greedy approximation for minimum connected dominating sets,” *Theoretical Computer Science*, vol. 329, (1-3), pp. 325-330, Dec. 2004.

[10] M.T. Thai, F. Wang, D. Liu, S. Zhu, and D.Z. Du, “Connected dominating sets in wireless networks with different transmission ranges,” *IEEE Trans. on Mobile Computing*, vol. 6, (7), July 2007.

[11] M.T. Thai, N. Zhang, R. Tiwari, and X. Xu, “On approximation algorithms of k-connected m-dominating sets in disk graphs,” *Theoretical Computer Science*, vol. 385, (1-3), pp. 49-59, 2007.

[12] M.T. Thai and D.-Z Du, “Connected dominating sets in disk graphs with bidirectional links,” *IEEE Communication Letters*, vol. 10, (3), pp. 138-140, 2006.

[13] F. Wang, M.T. Thai, and D.Z. Du, “On the construction of 2- connected virtual backbone in wireless network,” *IEEE Trans. on Wireless Communications*, vol. 8, (3), pp. 1230-1237, 2009

[14] X. Cheng, X. Huang, D. Li, W. Wu, and D.-Z Du, “Polynomial- time approximation scheme for minimum connected dominating set in ad hoc wireless networks,” *Networks*, vol. 42, (4), pp. 202-208, 2003.

[15] X. Cheng, M. Ding, D.H. Du, and X. Jia, “Virtual backbone construction in multi-hop ad hoc wireless networks,” *Wireless Communications and Mobile Computing*, vol. 6, (2), pp. 183-190, 2006.

[16] J. Blum, M. Ding, and X. Cheng, "Applications of connected dominating sets in wireless networks," in *Handbook of Combinatorial Optimization*, Birkhäuser: Kluwer Academic Publisher, 2004, pp. 329-369

[17] B. Das, R. Sivakumar, and V. Bharghavan, "Routing in Ad Hoc Networks Using a Spine*,"* in *International Conference on Computer Communications and Networks (ICCCN)*, Las Vegas, 1997, pp. 1–20

[18] J. Wu. "An extended dominating-set-based routing in ad hoc wireless networks with unidirectional links", *IEEE Trans. on Parallel and Distributed Systems*, vol. 13, (9), pp. 866 – 881, Sept. 2002.

[19] F. Dai and J. Wu. "An extended localized algorithm for connected dominating set formation in ad hoc wireless networks", *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, (10), pp. 902 – 920, Oct. 2004.

[20] J. Loughry, J. I. van Hemert, and L. Schoofs. (2000, December). Efficiently Enumerating the subsets of a set. [Preprint]. Available: http://www.applied-math.org/subset.pdf

[21] Deying Li, Hongwei Duc, Peng-Jun Wan, "Construction of strongly connected dominating sets in asymmetric multihop wireless networks", *Theoretical Computer Science*, vol. 410, (8-10), pp. 661-669, 2009.

[22] Hang T. Lau, "Java Library of Graph Algorithms and Optimization", in *Discrete Mathematics and its Applications*, New York: Taylor and Francis Group, LLC, 2007, pp. 383

Appendix A

Code for LDHD

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void RemoveVertex();
void SetBackVertex(int vertex);
void OutgoingDepthFirstSearch(int vertex, int * oVisitedCount);
void IngoingDepthFirstSearch(int vertex, int * iVisitedCount);
int isStronglyConnected();
int minDegreeVertex();
void initialize();

int N;
int E;
int* iDegree;
int* oDegree;
int** iAL;
int** oAL;
int* color; // -1 red, 0 white, 1 black.
int white;
int* iVisited;
int* oVisited;
int CurrentVerticesCount;

double startT,stopT,cumTime=0;


void initialize() // read a graph from a file and initialize all the variables.
{
	FILE* fp;
	int i,j;
	int v1,v2;

	fp=fopen("Graph.txt","r");
	if(fp==NULL)
	{
		printf("Error Openning\n");
		return ;
	}

	fscanf(fp,"%d %d",&N,&E);
```

```
	iAL = (int**)malloc(N*sizeof(int*));
	for( i=0 ; i<N ; i++)
		iAL [i] = (int*) malloc(N*sizeof(int));

	oAL = (int**)malloc(N*sizeof(int*));
	for( i=0 ; i<N ; i++)
		oAL [i] = (int*) malloc(N*sizeof(int));

	iDegree = (int*)malloc(N*sizeof(int));
	oDegree = (int*)malloc(N*sizeof(int));

	color = (int*)malloc(N*sizeof(int));
	iVisited = (int*)malloc(N*sizeof(int));
	oVisited = (int*)malloc(N*sizeof(int));

	for(i=0;i<N;i++)
		color[i]= 0;

	for(i=0;i<N;i++)
		iVisited[i]= -1;

	for(i=0;i<N;i++)
		oVisited[i]= -1;

	for(i=0;i<N;i++)
	{
		iDegree[i]=0;
		oDegree[i]=0;
	}

	for(i=0;i<E;i++)
	{
		fscanf(fp,"%d %d",&v1,&v2);

		oAL[v1][oDegree[v1]++]=v2;
		iAL[v2][iDegree[v2]++]=v1;
	}

	white = N;
	CurrentVerticesCount = N;

	fclose(fp);
}

int minDegreeVertex()
{
	int i;
	int index = -1;
```

```
	int min = 2*N;

	for(i=0;i<N;i++)
	{
		if(color[i] == 0 && (iDegree[i] + oDegree[i])< min)
		{
			min = iDegree[i] + oDegree[i];
			index = i;
		}
	}

	return index;
}

void RemoveVertex(int vertex)
{
	int i,j;

	for(i=0;i<iDegree[vertex];i++)
	{
		for(j=0;j<oDegree[iAL[vertex][i]];j++)
		{
			if(oAL[iAL[vertex][i]][j] == vertex)
			{
				oAL[iAL[vertex][i]][j] = oAL[iAL[vertex][i]][oDegree[iAL[vertex][i]]-1];
				oAL[iAL[vertex][i]][oDegree[iAL[vertex][i]]-1] = -1;
				oDegree[iAL[vertex][i]]--;
				break;
			}
		}
	}

	for(i=0;i<oDegree[vertex];i++)
	{
		for(j=0;j<iDegree[oAL[vertex][i]];j++)
		{
			if(iAL[oAL[vertex][i]][j] == vertex)
			{
				iAL[oAL[vertex][i]][j] = iAL[oAL[vertex][i]][iDegree[oAL[vertex][i]]-1];
				iAL[oAL[vertex][i]][iDegree[oAL[vertex][i]]-1] = -1;
				iDegree[oAL[vertex][i]]--;
				break;
			}
		}
	}
	color[vertex] = -1;
	white-- ;
	CurrentVerticesCount -- ;
```

```
}

int IsStronglyConnected()
{
	int i,j;
	int iVisitedCount = 0;
	int oVisitedCount = 0;
	int vertex;

	for(i=0;i<N;i++)
	{
		if(color[i] !=-1)
		{
			vertex = i;
			break;
		}
	}

	for(i=0;i<N;i++)
		oVisited[i] = -1;

	OutgoingDepthFirstSearch(vertex, &oVisitedCount);
	if(oVisitedCount == CurrentVerticesCount)
	{
		for(i=0;i<N;i++)
			iVisited[i] = -1;

		IngoingDepthFirstSearch(vertex, &iVisitedCount);

		if(iVisitedCount == CurrentVerticesCount)
			return 1;
	}
	return 0;
}

void IngoingDepthFirstSearch(int vertex, int * iVisitedCount)
{
	int i;

	iVisited[vertex] = 1;
	(*iVisitedCount)++;

	for(i=0;i<iDegree[vertex];i++)
	{
		if(color[iAL[vertex][i]] !=-1)
		{
			if(iVisited[iAL[vertex][i]] == -1)
				IngoingDepthFirstSearch(iAL[vertex][i], iVisitedCount);
```

```
                }
        }
}
void OutgoingDepthFirstSearch(int vertex, int* oVisitedCount)
{
        int i;

        oVisited[vertex] = 1;
        (*oVisitedCount)++;

        for(i=0;i<oDegree[vertex];i++)
        {
                if(color[oAL[vertex][i]] !=-1)
                {
                        if(oVisited[oAL[vertex][i]] == -1)
                                OutgoingDepthFirstSearch(oAL[vertex][i], oVisitedCount);
                }
        }
}

void SetBackVertex(int vertex)
{
        int i,j;

        for(i =0; i<iDegree[vertex]; i++)
        {
                oAL[iAL[vertex][i]][oDegree[iAL[vertex][i]]++] = vertex;
        }

        for(i =0; i<oDegree[vertex]; i++)
        {
                iAL[oAL[vertex][i]][iDegree[oAL[vertex][i]]++] = vertex;
        }

        color[vertex] = 0;
        white++;
        CurrentVerticesCount++;
}

int IsDominating()
{
        int i,j;

        for(i=0;i<N;i++)
        {
                if(color[i] == 1)
                        continue;
                for(j=0;j<oDegree[i];j++)
```

```
                {
                        if(color[oAL[i][j]]==1)
                                break;
                }
                if(j == oDegree[i])
                        return 0;
        }
        return 1;
}
int IsAbsorbant()
{
        int i,j;

        for(i=0;i<N;i++)
        {
                if(color[i] == 1)
                        continue;
                for(j=0;j<iDegree[i];j++)
                {
                        if(color[iAL[i][j]]==1)
                                break;
                }
                if(j == iDegree[i])
                        return 0;
        }
        return 1;
}
int main()
{
        int i,j;
        int vertex;
        int maxDeg = 0;
        int maxVertex;
        int r = 0, b = 0,w =0;

        initialize();

        startT = clock();

        while(white)
        {
                vertex = minDegreeVertex();
                RemoveVertex(vertex);
                if(!IsStronglyConnected())
                {
                        SetBackVertex(vertex);
                        color[vertex] = 1;
                        white--;
```

```
		}
		else
		{
			maxDeg = 0;
			for(i=0;i<iDegree[vertex];i++)
			{
				if(color[iAL[vertex][i]] == 1)
					break;

				if(color[iAL[vertex][i]] != -1)
				{
					if(iDegree[iAL[vertex][i]] + oDegree[iAL[vertex][i]] > maxDeg )
					{
						maxDeg = iDegree[iAL[vertex][i]] +
oDegree[iAL[vertex][i]];
						maxVertex = iAL[vertex][i];
					}
				}
				else
				{
					printf("error");
				}
			}
			if(i == iDegree[vertex])
			{
				if(color[maxVertex] != 0)
				{
					printf("error");
				}
				color[maxVertex] = 1;
				white--;
			}

			maxDeg = 0;
			for(i=0;i<oDegree[vertex];i++)
			{
				if(color[oAL[vertex][i]] == 1)
					break;

				if(color[oAL[vertex][i]] != -1)
				{
					if(iDegree[oAL[vertex][i]] + oDegree[oAL[vertex][i]] > maxDeg )
					{
						maxDeg = iDegree[oAL[vertex][i]] +
oDegree[oAL[vertex][i]];
						maxVertex = oAL[vertex][i];
					}
				}
```

```
                                else
                                {
                                        printf("error");
                                }
                        }
                        if(i == oDegree[vertex])
                        {
                                if(color[maxVertex] != 0)
                                {
                                        printf("error");
                                }
                                color[maxVertex] = 1;
                                white--;
                        }
                }
        }

        stopT = clock();
        printf("\nTime taken: %f secs\n\n",(stopT-startT)/CLOCKS_PER_SEC);

        for(i=0;i<N;i++)
        {
                if(color[i] == 1)
                        printf("%d ",i);
        }

        for(i=0;i<N;i++)
                if(color[i] == 1)
                        b++;
                else if(color[i] == 0)
                        w++;
                else
                        r++;

                printf("\n\nb = %d , w = %d, r = %d\n",b,w,r);

        if(IsDominating())
                printf("\n\nThe Graph is:\nDominating\n");
        if(IsAbsorbant())
                printf("Absorbant\n");
        if(IsStronglyConnected())
                printf("Strongly Connected\n");
        getch();

}
```

Appendix B

Code for RANDOM GENERATOR III

RANDOM GENERATOR III

```
import java.io.*;
public class GenerateDiscGraphs {

    public static void main(String[] args) {

    int found=0;
    int count=0;
    while(found ==0 && count<100)
    {
    count++;
    int N=100;
    int []Node;
    Node nodes[]=new Node[N];
    int edgei[] = new int[(N*(N-1))];
    int edgej[] = new int[(N*(N-1))];
    int e=0;

    for(int i=0; i< N; i++)
    {
      nodes[i]= new Node();
    }
    for(int i=0;i< N; i++)
      for(int j=0;j<N;j++)
      {
            if(j==i)
                  continue;
            else
                  if (nodes[i].IsConnectedTo(nodes[j]))
                {
                  edgei[e]=i;
                  edgej[e]=j;
                  e++;
            }
      }
      CheckIfStronglyConnected C = new
CheckIfStronglyConnected(N,e,edgei,edgej);
      found= C.IsStronglyConnected();
      if(found==1)
      {
            System.out.println("The edge list is:\n : The number of edges is
"+ e);
        for(int i=0;i<e;i++)
            System.out.println(edgei[i] + "," + edgej[i]);
```

```
            try {
            FileWriter f=new FileWriter(new File("Graph.txt"));
            BufferedWriter bufferedWriter = new BufferedWriter(f);
            PrintWriter printWriter = new PrintWriter(bufferedWriter);

            printWriter.println(N);
            printWriter.println(e);
            for(int i=0;i<e;i++)
            {
            printWriter.print(edgei[i]);
            printWriter.print(" ");
            printWriter.println(edgej[i]);
            }

            printWriter.close();
            bufferedWriter.close();
            } catch (Exception ec) {
            ec.printStackTrace();
            }

      }
    }
    if(count == 100)
      System.out.println("Could not generate a strongly connected disc graph
after " + count + " trials.");
    else
      System.out.println("Succesfully generated a strongly connected disc
graph after " + count + " trial(s).");
   /*
      //print nodes
      for(int i=0; i<N; i++)
      {
            System.out.println(nodes[i]);
      }
    */
      //print edges

    }
}



import java.util.Random;
import java.lang.Math;
class Node {

      Random rand = new Random();
      int rmax=1200,rmin=1200;
      int Area=1200;

      private int xCoordinate,yCoordinate,r;


      public Node()
      {
            r = rand.nextInt(rmax-rmin+1) + rmin;
```

```
            xCoordinate = rand.nextInt(Area+1);
            yCoordinate = rand.nextInt(Area+1);
      }
      public boolean IsConnectedTo(Node n2)
      {
            double distance=Math.sqrt((Math.pow(n2.xCoordinate-
this.xCoordinate,2))+(Math.pow(n2.yCoordinate-this.yCoordinate,2)));
            if(distance <=this.r)
                  return true;
            else return false;
      }
      public String toString()

      {
            String data;

            data="The Node has:\nTransmission ratio of:" + r +  "\nX-
coordinate: " + xCoordinate + "\nY-coordinate: " + yCoordinate + "\n";

            return data;
      }

}
```