GENERATING QUALITY DOMINATING SET

by

Khursheed Mohammed

Bachelor of Science
Jawaharlal Nehru Technological University, India
May 2002

A thesis submitted in fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2005
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 1428574
Copyright 2005 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Thesis Approval
The Graduate College
University of Nevada, Las Vegas

JANUARY 6TH 2005

The Thesis prepared by

KHURSHEED MOHAMMED

Entitled

GENERATING QUALITY DOMINATING SET

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

1817-53
ABSTRACT

Generating Quality Dominating Set
By

Khursheed Mohammed

Dr. Laxmi P. Gewali, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

Construction of a small size dominating set is a well known problem in graph theory and sensor networks. A Connected dominating set (CDS) can be used as a backbone structure in sensor networks for message delivery and broadcast. The general dominating set problem is known to be NP-hard and some approximation algorithms have been proposed.

In most approximation algorithms for constructing connected dominating set only the size of the dominating set has been considered. In this thesis we address the problem of constructing connected dominating sets with several quality factors that include (i) diameter, (ii) risk-factor, and (iii) interference. We propose algorithms for constructing CDS of small diameter, reduced risk-factor, and reduced interference. We also report on the experimental investigation of the proposed techniques. Experimental results show that the proposed algorithms are very effective in reducing interference without significantly increasing CDS size. The proposed algorithms are the first algorithms in the sensor network community that address both size and interference for designing dominating sets.
TABLE OF CONTENTS

ABSTRACT ........................................................................................ iii

LIST OF FIGURES ........................................................................... v

ACKNOWLEDGEMENTS .................................................................... vii

CHAPTER 1 INTRODUCTION ......................................................... 1

CHAPTER 2 PRELIMINARIES AND LITERATURE REVIEW .......... 4
  2.1: Preliminaries ........................................................................ 4
  2.2: Centralized Algorithms for Connected Dominating Sets .... 5
  2.3 Localized Algorithms ............................................................ 12

CHAPTER 3 ON GENERATING QUALITY DOMINATING SETS .......... 15
  3.1 Introduction ......................................................................... 15
  3.2 Diameter Issue ...................................................................... 15
  3.3 Risk Factor .......................................................................... 19
  3.4 Interference and Dominating Set ......................................... 21
  3.5 Development of Sprinkler Algorithm .................................. 26

CHAPTER 4 IMPLEMENTATION AND PERFORMANCE RESULTS .... 31
  4.1 Introduction ......................................................................... 31
  4.2 Guha and Khuller Algorithm (GK) ....................................... 32
  4.3 Implementation of GK Algorithm (GK) ............................... 33
  4.4 Implementation of Reduced Diameter CDS (RD-CDS) ....... 35
  4.5 Implementation of Reduced Interference CDS (RI-CDS) ....... 36
  4.6 Implementation of Minimum Interference Tree (MIT) .......... 37
  4.7 Implementation of Sprinkler Algorithm .............................. 39
  4.8 Performance Results ............................................................ 39

CHAPTER 5 CONCLUSION AND FUTURE EXTENSIONS .......... 49

REFERENCES .................................................................................. 51

VITA ............................................................................................... 52
LIST OF TABLES

Table 4.1  CDS size obtained by various algorithms .............................................. 41
Table 4.2  Radius (in terms of hop count) obtained by various algorithms ............... 42
Table 4.3  Average risk-factor obtained by various algorithms ............................... 42
Table 4.4  Average interference obtained by various algorithms ............................. 43
Table 4.5  Maximum interference obtained by various algorithms ............................ 44
ACKNOWLEDGEMENTS

It is a great pleasure to thank the many people whose help and suggestions were so valuable in completing this thesis. My greatest acknowledgement is extended to my advisor, Dr. Laxmi P. Gewali, for his support, encouragement and patience. Without his help, so generously given, the completion of this thesis would just not have been possible.

I would like to thank Dr. Ajoy K. Datta, Dr. John T. Minor and Dr. Venkatesan Muthukumar for their direct and indirect contribution throughout this investigation.

I would also like to acknowledge with appreciation, the support provided by my colleagues Sridhar Veravalli and Xiaojun Liang. I especially would like to thank my parents and friends for listening to me when I needed their help.
CHAPTER 1

INTRODUCTION

Sensor nodes are small scale equipments embedded with low power devices that include a microprocessor, sensor, and wireless communication component. The embedded wireless devices have transmission capability up to 200 meters, operate in the frequency 2.4 GHz, and can transmit/receive voice and data streams. Bluetooth and IEEE 802.11 are the commonly used radio devices in sensor nodes. A sensor node is energized by a limited power battery source and can be either in active or inactive state. Any small computing elements such as laptop computers, handheld computers, cell phones, PDAs, and small scientific instruments can be enhanced to make them sensor nodes.

Sensor nodes can be connected into a network by using their radio links. There is no fixed infrastructure to connect them. The connectivity achieved by using radio links can change dynamically when the sensor nodes change their state from active to inactive and vice versa. Additional problems arise when sensor nodes are allowed to move. For these reasons a sensor network is often called an ad-hoc network. It is generally assumed that a sensor node knows the position of itself and the position of other nodes that lie within the transmission range. A sensor node can read the coordinates of its location by using small scale GPS devices embedded in it.

Sensor networks can be used in several application areas, which include disaster rescue, wireless conferences, battlefield, object monitoring in remote and/or dangerous
environment, wireless surveillance, traffic monitoring, and wireless internet. Development of practical protocols/algorithms for solving communication related problems in an ad-hoc network is very challenging and requires radically different approaches than the ones used in a traditional fixed wired network. Due to limited computing resources, centralized algorithms needing global knowledge of the network may not be feasible in ad-hoc sensor network. Localized and distributed algorithms that can be executed co-operatively from each node by using local network information are preferred in sensor network applications.

A sensor node can communicate directly with other nodes located within the transmission radius of its radio link. Two nodes not within transmission range can communicate by using other nodes as intermediate relay nodes. Routing, broadcasting, cluster formation, topology control, and power aware scheduling, are some of the primary research areas in algorithmic sensor networks. **Routing** is the process of arranging a sequence of nodes between a pair of source and destination nodes, so that a message can be propagated from the source node to the destination node by using nodes in the sequence as the intermediate relay nodes. **Broadcasting** is the process of sending a message from a source node to all other nodes in the network. An easy way to broadcast is to use simple **flooding**. In flooding, a node sends newly received messages to all its neighbors. Flooding has some serious problems that include network contention, power waste, collision, and resource misuse [9]. Cluster based forwarding and geographic forwarding can be used to improve upon the performance of broadcasting algorithms [8].

The concept of a dominating set from graph theory is very useful in generating a backbone network of reduced size for the underlying ad-hoc sensor network. The general
problem of constructing a minimum size dominating set is known to be NP-hard [4] and some centralized, distributed, and localized algorithms for generating approximate solutions have been reported [4]. None of these algorithms consider interference in generating connected dominating sets. In this thesis we address the problem of generating connected dominating sets while taking interference into account.

In Chapter 2 we present a critical review of important existing algorithms for generating a connected dominating set (CDS). In Chapter 3, we highlight the importance of an interference factor in developing dominating set algorithms. We then present two centralized algorithms for generating CDS. The first one, referred to as the Reduced Diameter - Connected Dominating Set (RD-CDS) algorithm, generates a CDS of small size and reduced diameter in $O(n^3)$ time, where $n$ is the number of nodes in the network. The second algorithm, called the Sprinkler algorithm, generates CDS starting from the construction of a minimum interference tree. The Sprinkler algorithm runs in time $O(n^2 \log n)$. In Chapter 4, we present an experimental investigation of the proposed algorithms and the well known GK-algorithm. The proposed algorithms and the GK-algorithm are implemented in the Java programming language. The implementation has a friendly interface that allows the user to enter randomly generated nodes and nodes selected by mouse clicks. Performance evaluations show that the proposed Sprinkler algorithm generates CDS of acceptable size and very small interference. Finally, in Chapter 5, we discuss new problem areas and extensions of the proposed algorithms.
CHAPTER 2

PRELIMINARIES AND LITERATURE REVIEW

2.1: Preliminaries

In this chapter we present preliminaries, definitions, and a review of literature related to the construction of connected dominating sets (CDS) with applications in sensor networks. Consider a distribution of \( n \) sensor nodes in the Euclidean space. The transmission ranges of all sensor nodes are assumed to be identical and equal to some constant \( r \). Without loss of generality we can take \( r \) as 1. The process of sending messages from a source node to a set of destination nodes is called \textit{multicasting}. Multicasting becomes \textit{broadcasting} if all nodes are destination nodes. For broadcasting, multicasting, routing, and related tasks, it is beneficial to construct a smaller connected network that represents the skeleton of the entire network. The skeleton network is to be designed in such a way that the nodes not present in the skeleton are within the transmission range of some nodes in the skeleton. For designing efficient algorithms for routing, broadcasting, covering and related problems, good quality skeleton networks are highly desired. Algorithms based on skeleton network tend to reduce overhead for access time and update time.

The notion of a skeleton network is closely related to the notion of a dominating set in graph theory [4]. Consider a connected graph \( G(V, E) \) with vertex set \( V \) and edge set \( E \). A sub-set of nodes \( R \subseteq V \) is called a \textit{dominating set} if any vertex in \( V \) is...
either in $R$ or is adjacent to a node in $R$. Even though the original graph is connected, the graph induced by the dominating set need not be connected. For applications in ad-hoc networks, it is necessary to have the connectivity property. A connected dominating set (CDS) is a dominating set whose induced graph is connected. Figure 2.1 illustrates these definitions.

A restricted class of graph called a Unit Disc Graph (UDG) has been commonly used to model a wireless ad-hoc network [8]. Formally, a unit disc graph $UDG (V, E)$ is a graph consisting of a set of $n$ sensor nodes $V = \{v_1, v_2, v_3..., v_n\}$. Two nodes $v_i$ and $v_j$ are connected by an edge if the Euclidean distance between them is less than 1. We can imagine a disc $D (v_i, v_j)$ with $v_i$ and $v_j$ as the end points of its diameter. Then $v_i$ and $v_j$ are connected by an edge if the diameter of the disc is less than 1. Figure 2.2 shows a unit disc graph and a connected dominating set.

![Connected Dominating Set (CDS) and Unit Disc Graph (UDG)](image)

**Figure 2.1: Connected Dominating Set (CDS)  Figure 2.2: Unit Disc Graph (UDG)**

2.2: Centralized Algorithms for Connected Dominating Sets

Several algorithms have been proposed [1, 4, 11] to find the dominating set and connected dominating set of a graph. The general problem of finding a dominating set of
smallest size is a very difficult problem. In fact the problem of finding the smallest connected dominating set for a graph is known to be NP-hard [4]. Some approximation algorithms for finding small size connected dominating sets (CDS) have been reported. One such centralized approximation algorithm that can yield solutions with approximation factor related to the maximum degree of the node in reported in [4]. Some distributed and localized algorithms for computing CDS for a connected graph have been reported recently [4].

Khuller and Guha [4] have proposed two centralized approximation algorithms for computing CDS. Their algorithms are based on the construction of a ‘special’ spanning tree of the given graph. The first algorithm, referred to as the basic algorithm, begins the construction of the spanning tree by selecting the largest degree node as the root of the partial tree. The tree is grown a few nodes at a time, by selecting the leaf node that maximizes the number of adjacent nodes outside the partial tree. The final connected dominating set is given by removing the leaf nodes from the constructed spanning tree.

This algorithm can be described more precisely in terms of a marking procedure. Initially, all vertices are unmarked (colored white). The largest degree node is marked black and is considered as the root of the spanning tree. Neighbors of the root node are colored gray. The tree formed by connecting gray nodes to the root node is the initial tree. The tree is grown by converting the carefully selected gray node to a black node. When a gray node is converted to a black node, unmarked (white colored) nodes adjacent to the newly colored black node are colored gray. The gray node that has the maximum number of white neighbors is selected to be colored black. This process of converting (i) one gray
node to black node and (ii) one or more white nodes to gray nodes is called scanning. Scanning is continued until all nodes are colored black or white.

| Input: A connected graph $G (V, E)$. |
| Output: Set of nodes $R \subseteq V$ that forms a connected dominating set of $G$. |
| **Step 1:** |
| • Let $v \in V$ be the largest degree node. |
| • Color $v$ black |
| • Include $v$ to empty tree $T$ |
| • For all nodes $w$ adjacent to $v$ { |
|   o Color $w$ gray |
|   o Add edge $(v, w)$ to $T$ |
| } |
| **Step 2:** |
| • while there is at least one white node { |
|   o Let $w$ be the gray node with the maximum number of white neighbors |
|   o Color $w$ black |
|   o For all white nodes $x$ adjacent to $w$ { |
|     ▪ Color $x$ gray |
|     ▪ Add edge $(w, x)$ to $T$ |
|   } //for |
| } //while |
| **Step 3:** |
| • $R$ is set to non-leaf nodes of $T$ |

Figure 2.3: GK-Basic Algorithm

At each step, black and gray nodes together make a tree whose internal nodes are colored black and leaf nodes are colored gray. The black nodes in the final tree give the connected dominating set. A formal sketch of the basic algorithm is listed above in Figure 2.3. Figure 2.4(a-h) illustrates a trace of the execution of the basic algorithm.
Figure 2.4: Execution Trace of GK Algorithm

- **a:** Given UDG
- **b:** Starting with the highest degree node
- **c:** Selecting highest degree gray node
- **d:** ...contd.
- **e:** ...contd.
- **f:** ...contd.
- **g:** ...contd.
- **h:** Selecting non leaf nodes
The basic algorithm can be improved by slightly modifying the scanning rule. A new operation called "scanning a pair of adjacent vertices" is introduced. While scanning at each step, a pair of adjacent vertices \( u \) and \( v \) is selected. Let \( u \) be gray and \( v \) be white. Scanning the pair means, first marking \( u \) black (makes more nodes gray) and then coloring \( v \) black (makes more nodes gray). The total number of nodes that are colored gray is called the "yield" of the scan step. Scanning a pair of vertices can be considered as a 'look-ahead' scan. At each step the algorithm performs an 'ordinary-scan' and a 'look-ahead' scan. The scan that has the larger yield is used to grow the tree. A formal sketch of the look-ahead algorithm is listed below in Figure 2.5 and an example execution trace is shown in Figure 2.6.
Input: A connected graph G (V, E).
Output: Set of nodes R ⊆ V that forms a connected dominating set of G.

Step 1:
- Let v ∈ V be the largest degree node.
- Color v black and include v to empty tree T
- For all nodes w adjacent to v
  - Color w gray and add edge (v, w) to T

Step 2:
- while there is at least one white node {
  - Let w be gray node
  - Let u, v be adjacent nodes with gray and white colors respectively
  - yield (w) = the number of white neighbors of gray node u
  - yield (u, v) = the total number of white neighbors to either to u or v excluding v
  - Find either w or u, v pair with maximum yield
  - If yield (u, v) > yield (w) {
    - //Scan u first
    - Color u black
    - For all white nodes x adjacent to u
      - Color x gray and add edge (u, x) to T
  - //and now scan v
  - Color v black
  - For all white nodes x adjacent to u
    - Color x gray and add edge (v, x) to T
  }
  Else {
    - //Scan w
    - Color w black
    - For all white nodes x adjacent to u
      - Color x gray and add edge (w, x) to T
  }
}

Step 3:
- R is set to non-leaf nodes of T

Figure 2.5: GK Look-Ahead Algorithm
Figure 2.6: Execution Trace of GK Look-Ahead Algorithm

a: Given UDG  
b: Selecting the highest degree node

c: ...contd.  
d: ...contd.

e: ...contd.  
f: ...contd.

g: Extraction of Skeleton
The two polynomial time algorithms give approximation factors of $2H(\Delta) + 2$ and $H(\Delta) + 2$. Where the $\Delta$ is the maximum degree and $H$ is the harmonic function.

2.3 Localized Algorithms

These centralized algorithms are not very attractive for constructing CDS, particularly in mobile computing and sensor network applications. Centralized algorithms need to have global network information for execution. Some researchers have proposed distributed algorithms for constructing CDS [1, 7]. Most distributed algorithms for CDS are theoretically interesting but are difficult for real world implementation. Some distributed algorithms for CDS are, in fact, distributed implementations of the variations of centralized algorithms and have high message complexity overhead [8].

In recent years, a few interesting localized algorithms for CDS have been proposed [7, 11]. Localized algorithms for CDS do not have a performance guarantee but generate acceptable results for most sensor node distributions. Simplicity of implementation and dependence only on local information are the two notable attractive features of localized algorithms for CDS. While one can construct rare node distribution where localized CDS algorithms may generate solutions of very large size, it has been found that for randomly distributed node sets the size of the generated solution is fairly acceptable [7, 11].

One of the first localized algorithms for generating CDS was proposed by Wu and Li [11]. Their algorithm marks nodes purely on the basis of 1-neighbor and 2-neighbor information. Each node examines the connectivity of its neighbors locally and marks itself as a dominating node if the connectivity information satisfies a certain
property. Specifically, a node \( x \) marks itself black (dominating) if it has two neighbors \( u \) and \( v \) which are not directly connected. The dominating set produced by using only a local marking process can have many redundant nodes. Let us consider a network as shown in Figure 2.7a. When the marking process is performed on this network a CDS (black nodes and thick edges), as shown in Figure 2.7b results.

![Diagram of network and marking process](image)

- **a:** Connected Network
- **b:** CDS after marking process
- **c:** CDS after Rule 1
- **d:** CDS after Rule 2

**Figure 2.7: Illustrating Execution of Marking**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Wu and Li also proposed two post processing steps for identifying a few redundant nodes in the dominating set produced by the marking process. The first post processing step (Rule 1) un-marks a marked node \( x \) if (i) all neighbors of \( x \) are also the neighbors of another marked node \( y \) and (ii) nodes \( x \) and \( y \) are also neighbors. Figure 2.7c shows the result of elimination of redundant dominating nodes by applying Rule 1. The second post processing step (Rule 2) un-marks a mark node \( x \) if all neighbors of \( x \) are also neighbors of two adjacent marked nodes \( u \) and \( v \). Basically, marked nodes \( u \) and \( v \) are the witness to show that marked node \( x \) is redundant. Figure 2.7d shows the dominating set after removing the redundant nodes by applying Rule 2 to nodes in Figure 2.7c. While applying Rule 1 and/or Rule 2, more than one set of nodes may satisfy the connectivity and coverage property. To resolve this problem, a priority based on node identity and/or node degree can be used [11]. Localized algorithms for generating CDS by using geographic location of nodes has been reported in [7]. The above localized algorithms [7, 11] produce acceptable size solutions for most distribution of nodes, but can guarantee. One can construct rare counter examples where the localized algorithms produce large size CDS.
CHAPTER 3

ON GENERATING QUALITY DOMINATING SETS

3.1 Introduction

For constructing a connected dominating set of an ad-hoc sensor network, most researchers have considered only the number of dominating nodes as the objective function. It is usually emphasized that the smaller the size of the dominating set the better the quality. As mentioned in Chapter 2, the general problem of finding the smallest connected dominating set is NP-hard [4], and some success has been achieved in developing approximation algorithms [4]. While much emphasis has been spent on the size of the dominating set, not much research has been reported on other aspects of the dominating set. In this Chapter we address the quality aspects of dominating sets other than the size of the set. The specific other qualities we consider include (i) the network diameter, (ii) risk-factor, and (iii) interference. We propose algorithms for generating connected dominating sets (CDS) having small diameter, low risk factor, and low interference.

3.2 Diameter Issue

The diameter of a connected network is the maximum distance in the shortest path between any two nodes of the network. The diameter could be measured either in
term of the Euclidean distance or in terms of the number of edges in the path (the hop count). A connected dominating set with large diameter often leads to an increase in the propagation error. For reliable message delivery, a connected dominating set of small diameter is certainly preferred. The well known GK-algorithm [4] produces a dominating set of relatively small size but the resulting tree could have very large diameter. Figure 3.1a shows a dominating set produced by the GK-algorithm with large diameter. But this network admits a smaller diameter CDS as shown in Figure 3.1b.

![Illustrating a Large Diameter Connected Dominating Set](image)

Figure 3.1: Illustrating a Large Diameter Connected Dominating Set

This observation leads us to look for the development of algorithms for generating connected dominant sets of small size and reduced diameter. The problem can be defined formally as follows.
Reduced Diameter - Connected Dominating set (RD-CDS) Problem:

Given: Set of sensor nodes, each with fixed transmission radius.

Question: Construct a connected dominant set of small diameter for the sensor network induced by the given sensor nodes.

Since the GK-algorithm constructs a tree rooted at a given node, it is tempting to modify the tree construction approach so that the diameter of the resulting tree does not become large. In the GK-algorithm the tree is constructed by growing it from the root and adding nodes to the partial tree one node at a time. The added node is selected that results in 'maximum yield'. (We recall that the yield of a node \(x\) is the number of non-tree nodes that are dominated by \(x\), i.e. adjacent to \(x\).) Among the several candidate nodes that can be added to the partial tree, the one that dominates the maximum number of new nodes is selected. The process of growing the partial tree purely on the value of the 'yield' can lead to a tree of large diameter.

In order to check the growth of the tree deeper and deeper we need to modify the algorithm to proceed in a breadth first manner mimicking the construction of the breadth first search (BFS) tree [3]. The algorithm starts the construction of the tree from a selected node as the root. The nodes are processed in the order of increasing hop distance from the root. In other words, nodes at hop distance \(i+1\) are processed only when the processing of nodes at hop distance \(i\) have been completed. Suppose that the nodes within hop distance \(i\) have been processed to include or exclude in the dominating set tree. To process the nodes at level \(i+1\) the algorithm computes the yield for all nodes at level \(i+1\) and picks the one that gives the maximum yield to add to the tree. This process of adding maximum yield nodes is repeated until all nodes at level \(i+1\) are processed. A formal
sketch of the algorithm is listed below in Figure 3.2 as Reduced Diameter - CDS Algorithm.

**Input:** Unit disc graph $G (V, E)$ representing the sensor network.

**Output:** Set of nodes $R \subseteq V$ connected as tree of small diameter.

**Step 1:**
- Integer $level = 0$
- Let $v \in V$ be the largest degree node.
- Color $v$ black
- Child Level of $v = level$
- $level = level + 1$
- Queue $Q = NULL$
- Include $v$ to empty tree $T$
- For all nodes $w$ adjacent to $v$:
  - Color $w$ gray and add edge $(v,w)$ to $T$
  - Child Level of $w = level$; add $w$ to $Q$

**Step 2:**
- while there is at least one element in $Q$
  - while there is at least one gray node in $Q$ with Child Level = $level$
    - Let $w$ be the gray node with Child Level = $level$ the maximum number of white neighbors
    - Color $w$ black
    - Delete $w$ from $Q$
    - For all white nodes $x$ adjacent to $w$
      - Color $x$ gray and add edge $(w,x)$ to $T$
      - Child Level of $x = level$; add $x$ to $Q$
  - Delete all gray nodes with Child Level = $level$ from $Q$
  - $level = level + 1$

**Step 3:**
- $R$ is set to non-leaf nodes of $T$

Figure 3.2: Reduced Diameter - CDS Algorithm
Theorem 3.1: The Reduced Diameter Connected Dominating Set algorithm takes $O(n^3)$ time, where $n$ is the number of nodes in the network.

Proof: Step 1 is bounded by the maximum degree of any node of the network which is $O(n)$. A node is inserted into the queue at most one time – hence the outer while loop in Step 2 executes at most $O(n)$ time. The while condition of the inner loop in Step 2 can be checked in $O(n)$ time by scanning the content of the queue. The total time to check white nodes by the for loop in Step 2 takes $O(n)$ time. Hence the total time for Step 2 is bounded by $O(n^3)$ which is also the dominating time for the whole algorithm. □

3.3 Risk Factor

We now address the risk factor of a connected dominating set. Consider a connected dominating set (CDS) such as a tree rooted at a given node $r$. Such a tree is used as a back bone network to broadcast messages from the root node $r$ to all other nodes. If one of the nodes (say $x$) in the CDS tree becomes inactive then all nodes reachable only through node $x$ get disconnected when the CDS is used for message broadcasting. Thus a good measure of the risk factor should reflect the extent of the vulnerability of the CDS when one of the members of the dominating set becomes inactive. For a CDS with a tree structure, when a node $x$ becomes inactive all nodes which are reachable only from $x$ or from its descendants are disconnected. Of course, when the root node itself becomes inactive then all nodes are not reachable. This leads us to model risk-factor of a CDS in terms of the risk associated with the children of the root node $r$. 
We can illustrate this with an example CDS-tree, shown in Figure 3.3, where the CDS-tree is drawn with thick black edges. The root node $r$ has four children $a$, $b$, $c$, and $d$. An inspection of this network shows that the number of nodes which are reachable only through node $a$ is 3 (including a itself). Similarly, the number of nodes only reachable through nodes $b$, $c$, and $d$ are 2, 6, and 13, respectively. This shows that in term of reachability, node $d$ is most significant. We can measure the risk factor either in terms of the maximum risk or in terms of the average risk associated with the children of the root node. Let $cov(x)$ denote the number of nodes that can be reached only through node $x$ when the CDS-tree is used to broadcast message from the root node $r$. 

Figure 3.3: Illustrating Vulnerability
Definition 3.1: The maximum risk factor of a CDS-tree $T$, denoted by $\text{max-risk} \ (T)$, is the maximum value of $\text{cov} \ (x)$ when $x$ is any child of root node $r$. i.e.

$$\text{max-risk} \ (T) = \max \ \{ \text{cov} \ (x) \}, \ x \text{ is a child of } r.$$ 

Definition 3.2: The average risk factor of a CDS-tree $T$, denoted by $\text{avg-risk} \ (T)$, is the average value of $\text{cov} \ (x)$ where $x$ is a child of root node $r$. i.e.

$$\text{avg-risk} \ (T) = (1/m) \sum \text{cov} \ (x)$$

Where $x$ and $m$ denote a child and the number of children of root node $r$.

In the example CDS-tree shown in Figure 3.3, the value of $\text{max-risk} \ (T)$ is 13 and the value of $\text{avg-risk}(T)$ is 6.

3.4 Interference and Dominating Set

When too many sensor nodes are within each others' transmission range, then the radio signals from them can interfere significantly to degrade the very authenticity of the communicated message. Designing a sensor network topology that reduces interference is an important problem in ubiquitous computing. However very few research results addressing the interference issue have been reported [2]. It was generally believed that a network with small degree nodes should reduce interference. However, it was recently observed that a low node degree network could have high interference [2]. Figure 3.4 shows a low-degree network with high interference [2].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Most algorithms for generating CDS incorporate a greedy rule for picking nodes. Since the main objective of the CDS problem is to seek a dominating set of small size, most members of the solution are located around closely clustered nodes. On the other hand, groups of clustered nodes tend to increase interference. Broadly speaking, a topology resulting from a small size CDS tends to have more interference and vice versa. Blindly seeking a small size CDS can directly lead to a high interference topology as well. It is therefore very important to develop a CDS that generates a small size solution without increasing the interference significantly — what is needed is an acceptable compromise between the size of the solution set and the corresponding interference.

We start with the formal definition of interference by following the model suggested in [2]. We will assume each node can adjust its transmission power between zero and the maximum power level. Ability of power adjustment at each node is very useful for saving total energy use in a network obtained by topology control algorithms.
(Nodes of the network with short length incident edges can adjust power level just enough to reach the adjacent nodes.) Consider a pair of nodes $u$ and $v$ separated by a distance less than the maximum transmission radius $r$. Let $D(u, |u, v|)$ denote the disc centered at $u$ and radius $|u, v|$. Signals from node $u$ can potentially affect (interfere) signals generated by nodes lying within the disc $D(u, |u, v|)$. Similarly, signals from node $v$ can affect signals on nodes lying in the disc $D(v, |v, u|)$. Formally, this model of a node influence region, where the interference corresponds to an edge, can be defined as follows, and Figure 3.5 illustrates the definition.

**Definition 3.4.1:** The interference corresponding to an edge $e = (u, v)$ denoted by $I(e)$ is given by the count of nodes in the union of the discs $D(u, |u, v|)$ and $D(v, |v, u|)$.

\[ I(e = (u, v)) = \text{Size}\{w\mid w \text{ is in } D(u, |u, v|)\} \cup \{w\mid w \text{ is in } D(v, |v, u|)\} \]

\[ I = 19 \]

Figure 3.5: Interference Region of an Edge
To generate a CDS-tree of small size and low interference we need to examine the yield and interference for candidate edges and nodes. We propose an algorithm that grows a partial CDS-tree by adding one edge at a time. The added edge is selected in such a way that it tends to have low interference and high yield.

An interference factor for each edge of the unit disc graph induced by the sensor nodes can be computed by using Definition 3.4.1. Interference for each edge can be computed before constructing the CDS-tree. The value of yield for a node $x$ depends on the number of non-tree nodes adjacent to $x$. So, the yield of a node should be computed on the fly as partial tree construction progresses. Consider a partial CDS-tree shown in Figure 3.6. In the Figure 3.6, the nodes of the partial CDS-tree are drawn black. Non-tree nodes within the range of the CDS-tree are referred to as fringe nodes which are drawn as empty circles. The edges connecting the nodes of the CDS-tree with the fringe nodes are referred to as bridge edges (drawn as dashed line segments in Figure 3.6).

![Figure 3.6: Illustrating Partial CDS-tree](image)
Let $Y(g_i)$ and $l(e_i)$ be the yield and interference factors corresponding to the fringe node $g_i$ and its bridge edge $e_i$, respectively. Consider the following rule for selecting an edge from among all bridge edges.

**Rule 1**: Pick the edge from the bridge edges that maximizes the ratio $Y(g_i)/l(e_i)$. Initially, the algorithm picks the edge $e = (a, b)$ that maximizes the ratio $(Y(a) + Y(b))/l(e)$. At each subsequent step, the algorithm adds edges by following Rule 1. When an edge is added to the partial CDS-tree, the yield of neighboring nodes changes and hence the algorithm needs to update the yield of those nodes accordingly (recall that the yield of a node is the number of neighboring nodes not belong to the partial CDS-tree). A formal description of the algorithm is listed in Figure 3.7 as Reduced Interference - CDS Algorithm (RI-CDS).

| Input: A connected unit disc graph $G (V, E)$ representing a sensor network. |
| Output: A CDS-tree of small size and reduced interference. |

**Step 1:**
- For each edge $e \in E$ find $l(e)$

**Step 2:**
- Let $e' = (a, b)$ be the edge that maximizes the value $(Y(a) + Y(b))/l(e)$
- Tree $T = e'$
- Tree $T' = T \cup \{ e | e$ is a fringe edge of $T \}$

**Step 3:**
- While ($T'$ does not contain all nodes){
  - Compute the yields $Y(i)$ for all fringe vertices
  - Find the fringe edge $e_i = (a_i, g_i)$ that maximizes the ratio $Y(g_i)/l(e_i)$
  - $T = T \cup \{ e_i \}$

**Step 4:**
- Report $T$ as the dominating set

*Figure 3.7: RI-CDS Algorithm*
3.5 Development of Sprinkler Algorithm

Consider the edges of the unit disc graph (UDG) induced by \( n \) sensor nodes. Some edges are less prone to interference than others. The edges of UDG lying near the boundary of the convex hull of the sensor nodes are more likely to have less interference than the other edges. We can consider the half planes induced by the line passing through an edge \( e \) on the convex hull boundary. One of the half planes does not contain any sensor node and hence the interference at an boundary edge is very likely less than the interference at an interior edge. This is stated in the following observation.

**Observation 1:** The interference on a boundary edge is very likely less interference than at an internal edge.

Whenever we grow a tree rooted at a node in the interior of the convex hull, more internal and very few boundary edges are selected. This approach has the disadvantage of missing low interference edges in tree construction.

We can consider the structure of the minimum interference tree for the unit disk graph (UDG). The minimum interference tree of UDG is the minimum spanning tree where the weight of an edge \( e \) is the interference factor \( I(e) \) as stated in Definition 3.4.1. Figure 3.8 shows an example of a minimum interference tree (MIT). It is observed from this example that edges of MIT can cross.
A minimum interference tree for a set of $n$ sensor nodes can be computed in $O(n^2 \log n)$ time by using the algorithm reported in [2]. The algorithm is an adoption of the well know Kruskal's algorithm for a minimum spanning tree [3] by taking the interference corresponding to an edge as its weight.

If we take the minimum interference tree (MIT) as a basis for constructing a connected dominating set then we can get such a set by simply removing the edges incident on leaf nodes of MIT as shown in Figure 3.9. This shows that while the direct use of MIT yields a CDS-tree of minimum interference, the size could be prohibitively large. In order to reduce the number of dominating nodes, we process the MIT to identify additional nodes that are dominated by other nodes in MIT. We can pick a suitable node as the root and view MIT as a rooted-MIT.
Definition 3.4.2: Consider a pair of nodes \( x \) and \( y \) such that \( x \) is a parent of \( y \). If all descendents of \( y \) are within the transmission range from \( y \) but not from \( x \) then \( x \) is called a head node.

In Figure 3.9, take node \( r \) as the root. Then node \( a \) is a head node since all descendents of \( a \) are within the range from \( a \) but not within the range from \( p \). From each head node, descendent nodes that are within its range are connected directly and we call this as *sprinkler* formation. Figure 3.10 illustrates a sprinkler connection. Identification of head nodes and construction of sprinkler edges can be done recursively starting from the root node. The algorithm is listed in Figure 3.11 as *Sprinkler Tree Construction*. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Void MakeSprinkler(TreeNode t) {
  • Initially all nodes are colored WHITE
  • If (t == NULL) return
  • Else if (t is a leaf node) return
  • Else {//t has r subtrees t1, t2, t3, ..., tr
    o For (int i = 1; i<=r; i++) MakeSprinkler(t->childi)
    o Case 1: (the roots of all sub trees of t are BLACK)
      ▪ Color t BLACK; return;
    o Case 2: (the roots of all sub trees of t are WHITE)
      //Let tw1, tw2, tw3, ..., twi be the sub trees of t
      //with WHITE colored //root
      ▪ If (all nodes of t are within the range of t)
        o If (t is the root of MIT) color it BLACK; return;
      ▪ Else
        o For all sub trees ti that are not within the
          range of t
            //Make sprinkler
            ▪ Color t and root of twi BLACK
            ▪ Make direct connection from the
              root of twi to its descendents if necessary
        o Return
    o Case 3: (not case 1 and case 2)
      ▪ Color t BLACK;
      ▪ For all sub trees twi with WHITE root
        • If (nodes in twi are within the range of t)
          o Make direct connections from t to
            nodes in sub tree twi;
        • Else
          o Color root of twi BLACK
          o Make direct connections from root
            of twi to the nodes in twi; return;
  } //End of MakeSprinkler

Figure 3.11: Sprinkler Tree Construction

Lemma 3.1: MakeSprinkler function takes $O(n^2)$ time, where $n$ is the number of nodes in
the sensor network.

Proof: The recursive function is called exactly once from each node. The most expensive
operations corresponding to a node are (i) checking whether all its descendents nodes are
within the transmission range from the node, and (ii) making direct connections from the node to all descendents. Both of these can be done in \( O(n) \) time. Charging \( O(n) \) time for each node the total time for the function is \( O(n^2) \).

Given a sensor network with \( n \) nodes, the minimum interference tree (MIT) induced by the network can be constructed in \( O(n^2 \log n) \) time by using the algorithm suggested in [2]. From this MIT, a sprinkler tree can be constructed in \( O(n^2) \) time by invoking the MakeSprinkler() function. Hence we have the following theorem.

**Theorem 3.2:** Given a sensor network, a Sprinkler tree representing a CDS of small size and reduced interference can be constructed in \( O(n^2 \log n) \) time.
CHAPTER 4

IMPLEMENTATION AND PERFORMANCE RESULTS

4.1 Introduction

In this chapter, we mainly consider the implementation, experimental investigation and performance evaluation of the quality connected dominating set algorithms proposed in Chapter 3. The algorithms considered for experimental investigation are (i) Guha and Khuller (GK), (ii) Reduced Diameter Connected Dominating Set (RD-CDS), (iii) Reduced Interference Connected Dominating Set (RD-CDS), (iv) Minimum Interference Tree (MIT), and (v) Sprinkler Formation. All implementations have been done in the Java Programming language. Java supports an Object Oriented Programming facility, where real world objects can be represented using classes. We have used Java Swings extensively to do the necessary drawing. Java Swings has been chosen over Java AWT because it is a light-weight component and supports more sophisticated controls. To implement these algorithms, a Unit Disc Graph structure is considered as the basis which is imported from previous work done by Sridhar [10]. This graph is useful to determine whether a given set of nodes can form a connected graph for a given value of range $r$ and other properties. All the tree construction algorithms require a connected set of nodes for a given value of transmission range $r$. 

31

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
4.2 Guha and Khuller Algorithm (GK)

In Java, we represent a CDS tree structure as a class \textit{GKalg}. This class structure has all the necessary member variables and member methods to represent CDS trees. This class is inherited from class \textit{UnitDiscGraph}. The structure of class \textit{GKalg} is shown below.

\textbf{Class \textit{GKalg}}

\begin{itemize}
  \item \textit{Vector Vertices} \text{//This list holds all the vertices in the plane Z. Each Vertex is of type \textit{GKvertex Class}.}
  \item \textit{gkB\text{asicAlg}}() \text{//Computes CDS tree using \textit{GK Basic algorithm}.}
  \item \textit{gkLookA\text{headAlg}}() \text{//Computes CDS tree using \textit{GK Look Ahead algorithm}.}
  \item \textit{findFarthestNodeFromRoot}() \text{//Finds the farthest node from the root where distance is in terms of hop count.}
  \item \textit{GKvertex findRiskFactorForChildrenAtLevel(int lev)} \text{//Finds the average risk factor of nodes that are at a particular distance \textit{lev} in term of hop count from the root node.}
  \item \textit{int findInterference(GKvertex gk1, GKvertex gk2)} \text{//Finds the interference of two nodes \textit{gk1} and \textit{gk2}.}
  \item \textit{Clear()} \text{//Colors all nodes white.}
  \item \textit{scanRoot()} \text{//Finds the node with maximum degree and scans it.}
  \item \textit{Scan(GKvertex gk)} \text{//Scans the node \textit{gk}. Colors \textit{gk} black and the white neighbors of \textit{gk} as gray and adds them to the partial CDS tree.}
  \item \textit{boolean checkFlag()} \text{//Returns true if there exists at least one white node.}
  \item \textit{GKvertex findMaxYieldSingleNode()} \text{//Finds the node with maximum yield.}
\end{itemize}
4.3 Implementation of GK Algorithm (GK)

As mentioned earlier, we create an object of class GKalg to implement GK algorithm. The method gkBasicAlg is invoked on this object. Initially all nodes are colored white. The execution of the GK algorithm starts by finding a maximum degree node as the root and scanning (coloring it black and coloring the white neighbors as gray and adding them to the tree) starts from that node. In each subsequent step, the findMaxYieldSingleNode method finds a gray node (among all gray nodes) that gives maximum yield. This process of finding the maximum yield node and scanning is continued until all nodes are colored either black or gray. A pseudo-code version of the Java code of this method is shown below in Figure 4.1:
gkBasicAlg()
    Clear(); //Color all nodes as white.
    //Find the maximum degree node as root and scan it.
    scanRoot();
    //construct the CDS tree from the above partial tree
    while(checkFlag()) {
        GKvertex MaxYieldSingleNode = findMaxYieldSingleNode();
        //Scan it
        Scan(MaxYieldSingleNode);
    } //while
} //End of gkBasicAlg

Figure 4.1: Java-like Pseudo Code for GK-Basic Algorithm

Similar to gkBasisAlg method, the gkLookAheadAlg method initially colors all
nodes white and finds a node with maximum degree as root node and scans it. Up to this
stage both basic and look-ahead algorithms give the same partial CDS tree. While finding
the maximum yield node, the look-ahead algorithm looks one step ahead. It not only
considers the maximum yield of gray nodes but also the white nodes that are neighbor to
the gray nodes. In other words it finds a single gray node or a pair of adjacent gray and
white nodes, whichever gives the maximum yield. (Note that the yield of a gray and
white pair of nodes is the total number of white neighbors to either the gray or white pair
excluding the white node in the pair of consideration.) A pseudo-code version of the Java
code for this method is shown below in Figure 4.2:
gkLookAhead() {
    Clear(); //Color all nodes as white.
    //Find the maximum degree node as root and scan it.
    scanRoot();
    //construct the CDS tree from the above partial tree
    while(checkFlag()) {
        GKvertex SingleGrayNode = findMaxYieldSingleNode();
        GKvertex firstLevelWhiteNode = findMaxYieldTwoNodes();
        //scan which ever gives maximum yield
        if (MaxYieldValueForTwoNodes > MaxYieldValueForSingleNode) {
            Scan(firstLevelWhiteNode.Parent);
            Scan(firstLevelWhiteNode);
        } else {
            Scan(SingleGrayNode);
        }
    } //while
} //End of gkLookAhead

Figure 4.2: Java-like Pseudo Code for GK Look-Ahead Algorithm

4.4 Implementation of Reduced Diameter CDS (RD-CDS)

Similar to the GK algorithm, an object of class RD_CDS is created to implement this algorithm. In RD_CDS we proposed two algorithms (basic and look-ahead) similar to GK algorithm. The method RD_CDSbasic is invoked on this object. The two algorithms function on the similar concept of yield as done by GK algorithm, but the way the maximum yield nodes are selected is restricted to progress in a breadth first manner, so that the resulting CDS-tree lead to a small diameter CDS-tree. We measure the radius instead of the diameter which gives a better measure for reliability. Simply, the algorithm progresses level by level. The process of finding the maximum yield node and scanning it is done in the order of increasing hop distance from the root. In other words, nodes at hop distance \( i+1 \) are processed only when the processing of nodes at hop distance \( i \) have been completed. This enhances the reliability of the CDS-tree considerably. The method
RD_CDSbasic works in the same way as gkBascAlg except that it invokes findMaxYieldSingleNodeInQueue(level) instead of findMaxYieldSingleNode to find a node with maximum yield at a particular distance (in terms of hop count) level from the root node. When all nodes that have non-zero yield at a level have been scanned, the level is incremented and this process is continued until all nodes are colored either BLACK or GRAY.

4.5 Implementation of Reduced Interference CDS (RI-CDS)

As observed in Chapter 3, the GK algorithm picks the node having the highest yield but it also leads to high interference. In order to reduce interference and increase yield, we introduced a new criterion for selecting the nodes called yield-interference ratio. The yield-interference ratio of a gray node is the ratio of its yield to the interference factor of the edge joining it and its parent. The algorithm begins by selecting a pair of nodes a and b such that the yield-interference ratio of both a and b together is maximum, i.e., \(\frac{Y(a) + Y(b)}{I(ab)}\) is maximum. Among a and b, whichever has the maximum yield is taken as the root, and scanning proceeds. In the subsequent steps, the algorithm picks a node having the maximum yield-interference ratio.

Similar to RD-CDS algorithm, an object of class RI_CDS is created to implement this algorithm. The method RI_CDSalg is invoked on this object. Method findBaseEdge finds the root node and method findMaxYEdge finds the node having maximum yield-interference ratio. A Java like pseudo-code of this method is shown below in Figure 4.3:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
RI_CDSalg()
   Clear(); //Color all nodes as white.
   setInterferenceOfAllEdges(); //Calculates the interference of all edges
   //Initially find a pair of nodes which has maximum yield-interference ratio
   Edge baseEdge = findBaseEdge();
   //The end node that has maximum yield is selected as root
   if(findYield(baseEdge.GKI) > findYield(baseEdge.GK2))
   {
      Root = baseEdge.GKI;
      Scan(Root);
      //Scan the other end node only if it has WHITE neighbors
      if(findYield(baseEdge.GK2) > 0) Scan(baseEdge.GK2);
   }
   else
   {
      Root = baseEdge.GK2;
      Scan(Root);
      //Scan the other end node only if it has WHITE neighbors
      if(findYield(baseEdge.GKI) > 0) Scan(baseEdge.GKI);
   } //else
   //construct the rest of CDS tree from the above partial tree
   //grayNodes is a queue that holds the gray nodes which is updated in Scan
   //method
   while(!grayNodes.isEmpty()){
      Edge MaxYIedge = findMaxYIedge();
      //One end of MaxYIedge is GRAY and the other is BLACK
      //Scan the GRAY node
      if(MaxYIedge.GKI.Color == GRAY) Scan(MaxYIedge.GKI);
      else Scan(MaxYIedge.GK2);
      //remove the nodes that have zero yield from the grayNodes queue
      removeGrayNodesWithNoYieldFromQueue();
   } //while
} //End of RI_CDSalg

Figure 4.3: Java-like Pseudo Code for RI-CDS Algorithm

4.6 Implementation of Minimum Interference Tree (MIT)

In order to get a lower interference CDS-tree than given by RI-CDS, we implemented a Minimum Spanning Tree (MST) algorithm by using a variation of Kruskal's algorithm where the interference value is taken as the weight of the edges. We
call this special MST as the Minimum Interference Tree (MIT). The tree obtained by removing the edges incident on the leaf nodes is the CDS-tree.

The same object of class \textit{RljCDS} is used to implement this algorithm. The method \textit{MIT} is invoked on this object. A Java like pseudo-code of this method is shown below in Figure 4.4:

\begin{verbatim}
MIT (){
    //Calculate the interference of all edges and store in a global array
    //sortedEdgeListArray
    setInterferenceOfAllEdges();
    //Add an infinite interference edge at the end
    Edge nullEdge = new Edge(NullNode, NullNode, MAX_SIZE);
    sortedEdgeListArray[last+1] = nullEdge;
    //Sort the sortedEdgeListArray in non decreasing order of interference
    quickSort(0, last);
    int numberOfEdgesInMST =0; //Count the number of edges in the MIT so far
    int index =0;
    //While number of edges in MIT < (number of nodes - 1) AND
    //all edges in sortedEdgeListArray have been visited
    while(numberOfEdgesInMST < Vertices.size() && index<=last){
        //pick the next min-interference edge
        Edge e = (Edge) sortedEdgeListArray[index];
        //if it creates a cycle then discard it and move to next edge
        if (checkGroup(e) == true) {
            index++;
            continue;
        } //if
        //if no cycles
        else{
            MSTedges.add(e);
            index++;
            numberOfEdgesInMST++;
        } //else
    } //while
} //End of MIT
\end{verbatim}

Figure 4.4: Java-like Pseudo Code for MIT Algorithm
The \textit{MST} edges vector contains all edges of MIT. We invoke the method \textit{createTreeStructureForMST} to create the tree structure and stores the CDS nodes into a \textit{CDSnodes} vector.

4.7 Implementation of Sprinkler Algorithm

The Sprinkler algorithm uses the advantage of the MIT which produces a CDS-tree with low interference. It processes the MIT to identify additional nodes that are dominated by other nodes in the MIT. We can pick a suitable node as the root and view the MIT as a rooted-MIT. We picked the topmost node (lowest y-coordinate) in the CDS-tree formed by the MIT as the root $t$. The same object of class \textit{RIjCDS} is used to implement this algorithm. The method \textit{makeSprinkler (GKvertex r)} is invoked on this object by passing the argument root node $t$.

4.8 Performance Results

To evaluate the performance of the proposed algorithms, several connected dominating sets (CDS) were constructed by executing these algorithms on various randomly generated connected networks. The unit disc graph (UDG) induced by randomly generated nodes were taken as the input connected sensor network. The nodes were generated by randomly picking x- and y-coordinates in the range 20-650. These randomly generated co-ordinates were used to place nodes in a canvas of pixel size approximately 700 X 600. Several UDGs of node sizes in the range $n = 13$-415 were considered. The value of transmission radius was chosen appropriately to keep the UDG connected. For coordinates in the transmission range 80-200 and node size $n$ in the range
13-415, our preliminary experiments revealed that a transmission in the range 100-200 keeps the induced UDG connected. Specifically, we selected 150 as the value for the transmission radius to construct UDG.

Ten UDGs were randomly generated for a given values of node density (number of nodes per unit area). For each randomly generated UDG, CDSs were constructed by using the three proposed algorithms (RD-CDS, RI-CDS, and Sprinkler). For comparison and reference, CDS were also generated by using the well known GK algorithm [4] and recently reported MIT algorithm [2]. Values for maximum interference, average interference, size of the CDS, average risk-factor, and radius were measured. Tables 4.1-4.3 show the average values of different parameters for node densities 5, 10, 15 ...50.

The relationship between node density and CDS parameters such as interference, node size, etc. are plotted and shown in Figures 4.5 – 4.8. A few snap-shots of the output generated by the proposed algorithms as shown in Figure 4.9-4.21. By inspecting the plots, the main characteristics revealed from the experimental investigation can be summarized as follows:

- The size of the CDS produced by the Sprinkler algorithm is much better than that produced by the MIT algorithm. The size of the CDS produced by the Sprinkler algorithm is not as small as that produced by the well known GK algorithm, but is more or less in the middle between the size produced by the MIT and GK algorithms.

- For interference, the performance of Sprinkler and RI-CDS algorithms is much better than the performance of the GK algorithm. In fact, the sprinkler algorithm
produces CDS with minimum interference almost close to the optimum minimum value (as produced by MIT algorithm).

- The values of average interference and maximum interference rises up to node density 5 and then stays more or less flat beyond that for the CDS produced by the Sprinkler algorithm. The plateau shape of interference versus node density curve produced by the Sprinkler algorithm can be explained in term of the interference model. When the node density is low the lengths of the edges of the CDS-tree are large and consequently the dumb-bell area can enclose large number of other nodes and the corresponding interference value becomes high. When the node density is high, the edges of the CDS-tree have much smaller lengths and the dumb-bell shape contain very few other nodes and the resulting interference values is small and stays more or less constant.

- The Sprinkler algorithm is very effective in generating CDS-tree with near optimum minimum interference value and acceptable number of dominating nodes.

Table 4.1: CDS size obtained by various algorithms

<table>
<thead>
<tr>
<th>Node Density</th>
<th>GK</th>
<th>RD-CDS</th>
<th>RI-CDS</th>
<th>MIT</th>
<th>Sprinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.4</td>
<td>3</td>
<td>4</td>
<td>9.8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>5.6</td>
<td>6</td>
<td>6.6</td>
<td>15.4</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>10.6</td>
<td>10.8</td>
<td>11.8</td>
<td>26.4</td>
<td>22.4</td>
</tr>
<tr>
<td>10</td>
<td>10.8</td>
<td>11.6</td>
<td>13.6</td>
<td>54.2</td>
<td>40.2</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>12.8</td>
<td>16</td>
<td>83.4</td>
<td>52.2</td>
</tr>
<tr>
<td>20</td>
<td>11.4</td>
<td>13.4</td>
<td>16.2</td>
<td>108.2</td>
<td>64</td>
</tr>
<tr>
<td>25</td>
<td>11.4</td>
<td>13.2</td>
<td>17.4</td>
<td>136</td>
<td>68</td>
</tr>
<tr>
<td>30</td>
<td>11.8</td>
<td>14</td>
<td>18.8</td>
<td>165.6</td>
<td>81</td>
</tr>
<tr>
<td>35</td>
<td>12.2</td>
<td>13.8</td>
<td>21</td>
<td>193.2</td>
<td>105</td>
</tr>
<tr>
<td>40</td>
<td>12.2</td>
<td>13.8</td>
<td>20.6</td>
<td>220.6</td>
<td>108.4</td>
</tr>
<tr>
<td>45</td>
<td>11.2</td>
<td>14.2</td>
<td>21</td>
<td>245.6</td>
<td>114.8</td>
</tr>
<tr>
<td>50</td>
<td>12.4</td>
<td>13.6</td>
<td>20.2</td>
<td>273.4</td>
<td>117.2</td>
</tr>
</tbody>
</table>
Table 4.2: Radius (in terms of hop count) obtained by various algorithms

<table>
<thead>
<tr>
<th>Node Density</th>
<th>GK</th>
<th>RD-CDS</th>
<th>RI-CDS</th>
<th>MIT</th>
<th>Sprinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.2</td>
<td>1.2</td>
<td>2</td>
<td>6.6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2.6</td>
<td>2.2</td>
<td>2.6</td>
<td>9.8</td>
<td>8.8</td>
</tr>
<tr>
<td>5</td>
<td>5.2</td>
<td>4.8</td>
<td>6</td>
<td>15.6</td>
<td>15.2</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3.2</td>
<td>6</td>
<td>27.2</td>
<td>23.6</td>
</tr>
<tr>
<td>15</td>
<td>4.4</td>
<td>3</td>
<td>7</td>
<td>36.2</td>
<td>32.6</td>
</tr>
<tr>
<td>20</td>
<td>4.4</td>
<td>3.6</td>
<td>6</td>
<td>41.4</td>
<td>37.6</td>
</tr>
<tr>
<td>25</td>
<td>4.4</td>
<td>3.6</td>
<td>7.4</td>
<td>53.4</td>
<td>46.4</td>
</tr>
<tr>
<td>30</td>
<td>3.8</td>
<td>3</td>
<td>6.2</td>
<td>55</td>
<td>48.6</td>
</tr>
<tr>
<td>35</td>
<td>4.8</td>
<td>3.2</td>
<td>8.4</td>
<td>69.6</td>
<td>63</td>
</tr>
<tr>
<td>40</td>
<td>4.4</td>
<td>3.4</td>
<td>6.2</td>
<td>69.4</td>
<td>60.6</td>
</tr>
<tr>
<td>45</td>
<td>4</td>
<td>3</td>
<td>6.6</td>
<td>86.4</td>
<td>77</td>
</tr>
<tr>
<td>50</td>
<td>3.6</td>
<td>3</td>
<td>8</td>
<td>79.8</td>
<td>68.4</td>
</tr>
</tbody>
</table>

Table 4.3: Average risk-factor obtained by various algorithms

<table>
<thead>
<tr>
<th>Node Density</th>
<th>GK</th>
<th>RD-CDS</th>
<th>RI-CDS</th>
<th>MIT</th>
<th>Sprinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.4</td>
<td>0.4</td>
<td>2.4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4.4</td>
<td>3.7</td>
<td>4.266</td>
<td>9.5</td>
<td>9.5</td>
</tr>
<tr>
<td>5</td>
<td>15.466</td>
<td>9.932</td>
<td>15.4</td>
<td>21.434</td>
<td>22.4</td>
</tr>
<tr>
<td>10</td>
<td>14.466</td>
<td>7.3</td>
<td>19.45</td>
<td>43.9</td>
<td>50.1</td>
</tr>
<tr>
<td>15</td>
<td>21.934</td>
<td>6.9</td>
<td>26.15</td>
<td>53.26</td>
<td>67</td>
</tr>
<tr>
<td>20</td>
<td>19.166</td>
<td>9.94</td>
<td>48</td>
<td>102.6</td>
<td>100.4</td>
</tr>
<tr>
<td>25</td>
<td>33.168</td>
<td>12.94</td>
<td>59.134</td>
<td>116.866</td>
<td>152.2</td>
</tr>
<tr>
<td>30</td>
<td>24.9</td>
<td>10.608</td>
<td>47.1</td>
<td>157.6</td>
<td>148.4</td>
</tr>
<tr>
<td>35</td>
<td>43.816</td>
<td>14.51</td>
<td>87.866</td>
<td>108.366</td>
<td>167.3</td>
</tr>
<tr>
<td>40</td>
<td>39.432</td>
<td>23.506</td>
<td>58.332</td>
<td>208.666</td>
<td>223.6</td>
</tr>
<tr>
<td>45</td>
<td>38.98</td>
<td>15.538</td>
<td>69.834</td>
<td>214.4</td>
<td>238.8</td>
</tr>
<tr>
<td>50</td>
<td>31.95</td>
<td>21.37</td>
<td>125.934</td>
<td>162.566</td>
<td>252.8</td>
</tr>
</tbody>
</table>
Table 4.4: Average interference obtained by various algorithms

<table>
<thead>
<tr>
<th>Node Density</th>
<th>GK</th>
<th>RD-CDS</th>
<th>RI-CDS</th>
<th>MIT</th>
<th>Sprinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5.274</td>
<td>7.984</td>
<td>6.334</td>
<td>2.368</td>
<td>3.27</td>
</tr>
<tr>
<td>3</td>
<td>6.798</td>
<td>7.544</td>
<td>6.236</td>
<td>2.45</td>
<td>3.453</td>
</tr>
<tr>
<td>5</td>
<td>7.786</td>
<td>8.52</td>
<td>7.192</td>
<td>2.81</td>
<td>4.03</td>
</tr>
<tr>
<td>10</td>
<td>15.766</td>
<td>17.426</td>
<td>10.534</td>
<td>3</td>
<td>4.086</td>
</tr>
<tr>
<td>15</td>
<td>23.954</td>
<td>25.412</td>
<td>14.698</td>
<td>3.044</td>
<td>3.932</td>
</tr>
<tr>
<td>25</td>
<td>39.112</td>
<td>42.734</td>
<td>19.37</td>
<td>3.102</td>
<td>4.184</td>
</tr>
<tr>
<td>30</td>
<td>47.454</td>
<td>51.596</td>
<td>23.172</td>
<td>3.042</td>
<td>4.334</td>
</tr>
<tr>
<td>35</td>
<td>55.822</td>
<td>65.228</td>
<td>24.906</td>
<td>3.042</td>
<td>4.062</td>
</tr>
<tr>
<td>40</td>
<td>63.132</td>
<td>75.19</td>
<td>29.17</td>
<td>3.106</td>
<td>4.072</td>
</tr>
<tr>
<td>45</td>
<td>65.864</td>
<td>82.476</td>
<td>31.042</td>
<td>2.915</td>
<td>4.088</td>
</tr>
<tr>
<td>50</td>
<td>71.59</td>
<td>91.244</td>
<td>34.462</td>
<td>3.045</td>
<td>4.058</td>
</tr>
</tbody>
</table>

Table 4.5: Maximum interference obtained by various algorithms

<table>
<thead>
<tr>
<th>Node Density</th>
<th>GK Basic</th>
<th>RD-CDS</th>
<th>RI-CDS</th>
<th>MIT</th>
<th>Sprinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>9.2</td>
<td>9.2</td>
<td>8.4</td>
<td>4.6</td>
<td>4.4</td>
</tr>
<tr>
<td>3</td>
<td>11.6</td>
<td>11.6</td>
<td>10</td>
<td>5.6</td>
<td>5.6</td>
</tr>
<tr>
<td>5</td>
<td>14.2</td>
<td>13.8</td>
<td>13.2</td>
<td>7.6</td>
<td>8.2</td>
</tr>
<tr>
<td>10</td>
<td>28</td>
<td>28.4</td>
<td>21.6</td>
<td>8.2</td>
<td>8.2</td>
</tr>
<tr>
<td>15</td>
<td>40.8</td>
<td>40.2</td>
<td>30.2</td>
<td>9.2</td>
<td>8.2</td>
</tr>
<tr>
<td>20</td>
<td>51.2</td>
<td>53</td>
<td>36</td>
<td>9.6</td>
<td>8.6</td>
</tr>
<tr>
<td>25</td>
<td>66.6</td>
<td>64.2</td>
<td>54.2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>76.4</td>
<td>77.8</td>
<td>63</td>
<td>9.8</td>
<td>8.6</td>
</tr>
<tr>
<td>35</td>
<td>92.2</td>
<td>95.2</td>
<td>66.8</td>
<td>9.2</td>
<td>8.6</td>
</tr>
<tr>
<td>40</td>
<td>107.6</td>
<td>108.2</td>
<td>73.4</td>
<td>10</td>
<td>9.3</td>
</tr>
<tr>
<td>45</td>
<td>117.8</td>
<td>118</td>
<td>88.8</td>
<td>9.6</td>
<td>8.9</td>
</tr>
<tr>
<td>50</td>
<td>131</td>
<td>131.4</td>
<td>87.2</td>
<td>10.4</td>
<td>9.9</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 4.5: CDS Size Vs Node Density

Figure 4.6: Radius Vs Node Density

Figure 4.7: Average Risk-Factor Vs Node Density

Figure 4.8: Average Interference Vs Node Density

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 4.9: UDG of 112 nodes & Range 50

Figure 4.10: CDS-Tree for Figure 4.9 by GK-Basic Algorithm

Figure 4.11: CDS-Tree for Figure 4.9 by GK Look-Ahead Algorithm

Figure 4.12: CDS-Tree for Figure 4.9 by RD-CDS Algorithm
Figure 4.13: CDS-Tree for Figure 4.9 by RI-CDS Algorithm

Figure 4.14: CDS-Tree for Figure 4.9 by MIT algorithm

Figure 4.15: CDS-Tree for Figure 4.9 by Sprinkler Algorithm
Figure 4.20: CDS-Tree by MIT Algorithm

Figure 4.21: CDS-Tree by Sprinkler Algorithm
CHAPTER 5

CONCLUSION AND FUTURE EXTENSIONS

We presented a critical review of existing algorithms for constructing connected dominating sets. We also highlighted worst case scenarios. We proposed several algorithms namely Reduced Diameter Connected Dominating Set (RD-CDS), Reduced Interference Connected Dominating Set (RI-CDS), Minimum Interference Tree (MIT), and Sprinkler Formation to generate better quality connected dominating sets for sensor networks.

We also presented several experimental techniques needed to implement these algorithms. Implementation has been done in the Java programming language. The implementation prototype has a user friendly graphical user interface (GUI). The user can generate randomly distributed nodes which is adopted from the previous work done by Sridhar [10]. Each algorithm is displayed in a separate window to enable easy readability. Finally, the results are consolidated effectively to compare various algorithms.

To understand the performance of the proposed algorithms in specific terms we also implemented two other well known algorithms (GK algorithm and MIT algorithm) for generating CDS. The GK algorithm is designed for generating reduced size CDS, while the MIT algorithm is designed for minimizing interference. Two of our proposed algorithms (RI-CDS and Sprinkler) are designed to generated CDS-trees by reducing both size and interference factors. The Sprinkler algorithm is very effective in reducing
interference (just like but more than the MIT algorithm). The size of the CDS-tree generated by the Sprinkler algorithm is also much smaller than the size of the CDS-tree generated by the MIT algorithm. The RI-CDS algorithm is not as effective as the Sprinkler algorithm for reducing interference but is more effective in reducing size.

Node mobility is not considered in this investigation. In real situations, nodes may change their position with time. Nodes may also become inactive after a certain time and vice versa. It would be very interesting to develop algorithms for generating reliable CDS trees by integrating the RD-CDS, RI-CDS, and Sprinkler Formation approaches discussed earlier.
REFERENCES


6. Joseph O'Rourke, [http://cs.smith.edu/~orourke](http://cs.smith.edu/~orourke)


VITA

Graduate College
University of Nevada, Las Vegas

Khursheed Mohammed

Local Address:
4236, Chatham Circle, Apt #1
Las Vegas, NV – 89119.

Degree:
Bachelor of Technology, 2002
Jawaharlal Nehru Technological University, India.

Thesis Title:
Generating Quality Dominating Set

Thesis Examination Committee:
Chair: Dr. Laxmi P. Gewali, Ph. D.
Committee Member: Dr. John T. Minor, Ph. D.
Committee Member: Dr. Ajoy K. Datta, Ph. D.
Graduate Faculty Representative: Dr. Venkatesan Muthukumar, Ph. D.