ON ROUTING, BACKBONE FORMATION AND BARRIER COVERAGE IN WIRELESS AD HOC AND SENSOR NETWORKS

Mona Mehrandish

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy
Concordia University
Montréal, Québec, Canada

April 2011

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Mona Mehrandish**

Entitled: **On Routing, Backbone Formation and Barrier Coverage in Wireless Ad Hoc and Sensor Networks**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Wen-Fange Xie _____ Chair

Dr. Paola Flocchini _____ External Examiner

Dr. Chadi Assi _____ External to Program

Dr. Thomas Fevens _____ Examiner

Dr. Hovhannes Harutyunyan _____ Examiner

Dr. Lata Narayanan _____ Thesis Supervisor

Dr. Jaroslav Opatrny _____ Thesis Supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____ _____

Dr. Robin A. L. Drew, Dean
Faculty of Engineering and Computer Science

Abstract

On Routing, Backbone Formation and Barrier Coverage in Wireless Ad Hoc and Sensor Networks

Mona Mehrandish, Ph.D.
Concordia University, 2011

In this thesis, we provide some primitives for wireless ad hoc and sensor networks to facilitate applications such as disaster relief, community mesh networks, area monitoring and surveillance, and environmental monitoring. We introduce a local learning algorithm for routing that uses feedback from neighbors to avoid voids in the network. After five retrials, our algorithm achieves almost 100% delivery rate and has a stretch factor close to that of greedy routing. We then give an algorithm for the construction of a *connected dominating set* to serve as a data gathering and dissemination backbone in networks modeled by unit disk graphs as well as quasi unit disk graphs. We also consider the more general case of nodes with different transmission ranges modeled by disk graphs and we give an algorithm for the construction of a *strongly connected dominating and absorbent set*. Both our backbone construction algorithms are local and have constant approximation ratio. Through extensive simulations, we show that our algorithms outperform the existing algorithms for the same problems. Finally, we study problem of covering a barrier with mobile sensors in order to detect intruders as they cross the border of a protected area. We consider the problem of assigning final positions to sensors to provide maximum coverage of the barrier while minimizing the maximum movement of any sensor on both multiple line barriers as well as circular barriers. Furthermore, we consider the problem of minimizing the number of sensors moved so as to achieve maximum coverage of multiple line barriers as well as a circular barrier. For both our barrier coverage problems, we consider all possible scenarios depending on whether complete coverage is possible or not, and if complete coverage is not possible, whether the coverage is contiguous or non-contiguous. For each case, we either give efficient polynomial algorithms or we show the problem to be NP-hard.

Acknowledgments

It is a pleasure to thank those who made this thesis possible. I am heartily thankful to my supervisors, Lata Narayanan and Jaroslav Opatrny, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. Also, I would like to recognize many valuable contributions from my colleague Hossein Kassaei with whom I explored the CDS problem for UDGs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in wireless communications and electronics are paving the way for the deployment of low-cost, low-power networks of untethered and unattended sensors and actuators. Wireless ad hoc and sensor networks are infrastructureless networks formed by autonomous nodes communicating via radio interfaces.

A variety of applications such as disaster relief, community mesh networks, area monitoring, surveillance and environmental monitoring, and intrusion detection have been proposed for these networks. To enable the deployment of large scale ad hoc and sensor networks, a number of fundamental problems need to be solved in an efficient manner. Such problems include routing, broadcasting, topology control, data aggregation, backbone formation, and coverage of a given region or barrier. In this thesis, we consider the routing problem, backbone formation for data gathering and dissemination and barrier coverage for intrusion detection. In the following sections, we provide some brief background on each of the problems considered in this thesis.

1.1 Routing

Routing in wireless sensor networks (WSNs) is very challenging due to the inherent characteristics that distinguish these networks from other wireless networks like cellular networks. Due to energy restrictions of sensor nodes, it is not possible to build a global addressing scheme for the deployment of a large number of sensor nodes as the overhead of ID maintenance is high. Thus, traditional IP-based protocols may not be applicable to WSNs. Furthermore, sensor nodes that are deployed in an ad hoc manner need to be self-organizing as the ad hoc deployment of these nodes requires the system to form connections and cope with the resultant nodal distribution especially when the operation of the sensor networks is unattended. Additionally, sensor nodes are tightly constrained in terms of energy, processing, and storage capacities. Thus, they require careful resource management.

Since nodes not within the transmission range of each other cannot communicate directly, geometric proximity information has a high correlation with the network topology. Such an abstraction of the network connectivity based on nodes' Euclidean coordinates has tremendously simplified the design of routing protocols and improved routing efficiency. The class of routing algorithm in which each node along a source-destination path makes a message-forwarding decision based on some position information is called geometric routing [LW07]. For example, in geographical forwarding, forwarding decisions are based on the geographical locations of destinations and the one-hop (in some cases up to k-hop) neighborhood of a node. In a greedy manner, a packet is forwarded to the one-hop neighbor to make the most progress according to some metric such as Euclidean distance, angle with the source-destination line and etc [Fin87, KSU99, SSB99]. For a sensor network with uniform and dense sensor deployment in a flat and regular region, geographical forwarding is an efficient, scalable, and local scheme that produces almost shortest paths with very little overhead.

However, these greedy protocols would fail in presence of voids (an area where there are no sensor nodes) or when none of the neighboring nodes would make progress toward the destination. In situations where guaranteed delivery is critical, a recovery phase can be used to guarantee the delivery of the packets. Typically, the recovery phase uses the right hand rule to guarantee the delivery of the packets. Since greedy algorithms usually result in routes that are close to the shortest path, these algorithms use a combination of greedy and a variation of *face routing* also referred to as *planar graph routing* as their recovery process [BMSU99, KK00, KGKS05, KWZZ03]. These algorithms have been studied in networks usually modeled by a *unit disk graph* (UDG) or a *quasi unit disk graph* (QUDG). A network is modeled by UDG when all the nodes have the same transmission range. There is an edge between two nodes u and v in the graph, whenever the Euclidean distance between two nodes is less than the transmission range. A QUDG with parameters r and R $(0 < r < R)$ over a set of points in the plane is defined as follows. For any two points u and v, if the Euclidean distance between u and v is at most r, then $\{u, v\}$ is an edge in the graph, and if the Euclidean distance is in $(r, R]$, then edge $\{u, v\}$ may or may not exist.

The only known local approach to dealing with voids is face routing. However, face routing has two disadvantages: it requires planarization, which may not always be possible, and secondly, the routes produced by the face routing algorithm tend to be quite long. An alternative local approach for avoiding voids is investigated in this thesis.

1.2 Backbone Formation

The major task in all monitoring applications in WSNs such as air pollution monitoring, forest fires detection, machine health monitoring and landslide detection is to gather and disseminate the sensed data. The resource scarcity of WSNs requires

that the data gathering and dissemination backbone be as small as possible to reduce interference and possibly increase network throughput as well as deplete fewer nodes and thus prolonging network lifetime. A *minimum connected dominating set* MCDS is a good candidate for a routing as well as a data gathering and dissemination backbone. A MCDS S, is the smallest set of nodes in a network where every node in the network not in S, has a neighbor in S and the subgraph induced by nodes in S connected. However, the problem of finding an MCDS even for the case of UDGs is NP-hard [CCJ90]. Thus, there are many heuristics that have been proposed for the problem as well as some algorithms with constant approximation ratio [AWF02, DSW02, ACR07, PDDB05]. However, some of these heuristics have been shown to perform poorly in the worst case, and some of them are not local. The existence of a simple local algorithm with small approximation ratio that performs very well in practice remains to be investigated.

1.3 Barrier Coverage and Intrusion Detection

A major application category of WSNs consists of military and security-related applications, including perimeter surveillance, critical infrastructure protection, and country border control, to name a few. The goal is to effectively detect intruders that attempt to penetrate the region of interest. This type of coverage is referred to as *barrier coverage*, where the sensors form a barrier for the intruders. However since most of the time, the sensors are deployed arbitrarily around the perimeter of the region to be protected, complete barrier coverage is not always guaranteed.

The approaches to address the barrier coverage problem can be widely divided into two groups depending on the mobility of the sensor nodes used for the coverage. The first approach deals with the scenario in which once sensors are arbitrarily dispersed along the perimeter of the barrier they remain stationary. The studies in

this category deal with either estimating a density to cover the barrier with high probability once the nodes are arbitrarily dispersed or the question of whether the barrier is completely covered or not [CKL07, KLA05, BBSK07]. Clearly this approach uses way more sensors than actually needed for the barrier coverage. The second approach takes advantage of node mobility and once sensors have been deployed, it instructs them to move to final positions so as to achieve maximum coverage [BBH+08, CKK+10, CKK+09, SLX+10, WCLP06]. The goal here is to minimize the energy while maximizing the coverage. Two different aspects of energy minimization, namely minimizing the maximum movement and minimizing the sum of movements have been considered in the literature [BBH+08, CKK+10, CKK+09]. The problem of minimizing the maximum movement have been considered for only a single line barrier and a variation of the problem, where one sensor is assigned a predetermined position is NP-hard even for a line segment when nodes have different transmission ranges [CKK+09]. This is an area that has attracted a lot of interest in recent years and there are many interesting questions that remain unsolved so far.

1.4 Thesis Contributions

In this thesis, we first introduce a localized learning routing algorithm, that we call *river routing*, which learns about voids or obstacles during the course of the execution of the algorithm. In the beginning, the river routing algorithm follows the greedy path. When the algorithm reaches a local minimum, the node at which river routing failed sends some information about the direction of the obstacle to its neighbors and the next time the route diverts from the greedy path when it gets close to the obstacle area. In networks with small obstacles, the river routing algorithm has a delivery ratio of almost 100% and it generates a route close to the shortest path in the graph representing the network. This algorithm can be used in graphs with nodes that have

information about their geographical positions.

Furthermore, we give an algorithm to form a *connected dominating set* (CDS) that can serve as a data gathering and dissemination backbone. Our CDS approximation algorithm is local and it has a constant approximation ratio. We consider networks with equal range sensors, modeled by UDGs as well as graph with irregularities in sensor ranges, modeled by QUDGs. Our algorithm for the construction of a CDS is local with constant approximation ratio and through extensive simulations, we show that it outperforms all the existing CDS construction algorithms in the literature.

Moreover, we consider a more realistic scenario where sensor nodes might have different transmission ranges or due to energy constraint they might not use their full power. In this case, we model the graph with a *disk graph* (DG), in which there is a directional edge (u, v) from u to v, if the Euclidean distance between u and v is less than or equal to the transmission range of u. In case of directed graphs, the dominating set problem translates to the *dominating and absorbent set* (DAS) problem. Given a directed graph, a subset of nodes D is a dominating set for any vertex v not in D, there exists an incoming edge to v from D. Analogously, a set A is an *absorbent set* if for any vertex v' not in A, there exists an outgoing edge from v' to A . A vertex set is a DAS if it is both a dominating set and an absorbent set. A DAS is a *strongly connected dominating and absorbent set* (SCDAS) if the subgraph induced by nodes in the DAS is strongly connected. Our local algorithm for the construction of a SCDAS has a constant approximation ratio and we show that the SCDAS generated by our algorithm is smaller than that of the other SCDAS algorithms through simulations.

Also, we consider the barrier coverage and intrusion detection problem. We generalize the results for minimizing the maximum movement (*MinMax*) in [CKK$^+$09] to the case of multiple barriers as well as circular barriers. For identical range sensors we

provide centralized polynomial time algorithms for all possible scenarios depending on whether or not complete coverage is possible and when complete coverage is not possible whether or not the maximum coverage forms a contiguous interval or not. We also show that for non-identical sensor ranges the problem remains NP-hard for circular barriers.

Finally, we study a new aspect of minimizing the energy, referred to as *MinNum* for the barrier coverage problem, where we minimize the number of sensors that need to be moved in order to achieve maximum coverage. Although the MinNum problem has been mentioned before [DHM+09], it has never been considered for the barrier coverage problem. Minimizing the number of sensors moved can minimize the total energy especially when the energy needed to initiate a movement is significantly large. We address the MinNum problem on line barriers as well as circular barriers and we present centralized polynomial algorithms for identical range sensors. For the case of non-identical range sensors we show the problem to be NP-hard for some cases.

1.5 Outline of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we present the local learning river routing algorithm and we evaluate the performance of our algorithm through simulations. Chapter 3 introduces a local algorithm for the CDS construction in UDGs and QUDGS and the performance of the algorithm compared to the other state-of-the-art algorithms is studied through extensive simulations. These results appeared in [KMNO09, KMNO10]. We adapt the local CDS construction algorithm of Chapter 3 for construction of a local SCDAS for DGs in Chapter 4. We compare the SCDAS computation algorithm through simulations with the other algorithms for the SCDAS construction in the literature. In Chapter 5, the generalization of the MinMax problem in [CKK+09] is studied for multiple barriers as well as circular

barriers and several efficient polynomial algorithms are introduced. The MinNum problem for the barrier coverage is studied in Chapter 6 and our results are published in [MNO11]. We conclude with some directions for future research in Chapter 7.

Chapter 2

River Routing

In this chapter, we propose a new local learning routing algorithm, called *river routing* in a sensor network of location-aware nodes. The algorithm uses feedback from the other sensor nodes in order to fulfill its routing task.

In Section 4.1, we discuss the related work on geographic routing in *wireless sensor networks*, WSNs. The river routing algorithm is presented in Section 2.2. The performance of our algorithm is evaluated through simulations in Section 2.3, followed by the concluding remarks in Section 2.4.

2.1 Literature Review on Geographic Routing in Sensor Networks

The most trivial approach to routing is flooding where a node, upon reception of a message forwards it to all its neighbors, unless it has heard the message before. Clearly, if there is a path between two nodes u and v, flooding guarantees delivery. However, this method is inefficient and expensive. Thus, several routing algorithms for WSNs and ad hoc networks have been developed. The first routing algorithms for ad

hoc networks followed the traditional approach of topology-based routing [ASSC02]. They can be categorized as table-driven protocols or demand-driven protocols. Table-driven routing protocols maintain up-to-date routing information between every pair of nodes. The changes to the topology are maintained by propagating updates of the topology throughout the network. Source-initiated on-demand routing creates routes only when desired by the source node. At this time a *route discovery* process is initiated within the network. Since information about paths is maintained in these protocols, a topology change possibly requires distant nodes to change their routing tables [ASSC02, AY05].

Several novel geographic routing algorithms have been proposed that allow routers to be nearly stateless since packet forwarding is achieved by using information about the position of candidate nodes in the vicinity and the position of the destination node only [GSB03, MH01]. These protocols select the next-hop towards the destination based on the known position of the neighbors and the destination. The position of the destination may denote the centroid of a region or the exact position of a specific node. In these algorithm the next node is always selected according to a heuristic usually to make progress toward the destination [LS98, Fin87, KSU99]. Location-based routing protocols can avoid the communication overhead caused by flooding, but the local minimum problem is common for most decentralized location-based routing protocols. A local minimum happens at node u, when none of the neighbors of node u makes progress toward the destination compared to node u. In order to circumvent this problem, several routing techniques have been proposed. These techniques usually include a greedy approach with a recovery phase when the greedy part fails. The recovery process is usually a variation of face routing or also referred to as planar graph routing as their recovery process [BMSU99, KK00, KGKS05, KWZ03, KWZZ03].

In [BMSU99], the authors introduced an algorithm referred to as *GFG* which

was a combination of the greedy distance based algorithm *GEDIR* in [LS98] and a recovery phase called *FACE_2*. GFG needs a preprocessing phase in which a planar subgraph of the graph representing the network is extracted. Then, it uses GEDIR to find the path toward the destination. If GEDIR fails at node u, it then uses FACE_2 until it finds a vertex v which is closer to the destination than v and then it starts GEDIR from v. The recovery phase FACE_2 is simply a variation of the Compass Routing II in [KSU99]. The basic idea in Compass Routing II is to use the right-hand rule to traverse the faces in the graph, intersecting the line connecting the source to the destination.

Most non-backbone based algorithms guaranteeing delivery follow a variation of the greedy algorithm and a variation of the Compass Routing II in [KSU99] in the recovery phase [BMSU99, KK00, KGKS05, KWZZ03, KWZ03]. All these algorithms need to extract a planar subgraph of the underlying graph before they can proceed with their recovery phases. However, in some cases extracting a planar subgraph might not be possible.

In this chapter, we present a routing algorithm, river routing, which takes advantage of the negative feedback from nodes each time it fails. Our algorithm does not need a planar graph. To the best of our knowledge, none of the studies except one have considered routing using negative feedback. The only exception is the study of Yu et al. [YEG01]. In [YEG01], a routing algorithm called GEAR was presented which discussed the use of geographic information while disseminating queries to appropriate regions since data queries often include geographic attributes. The protocol used energy-aware and geographically-informed neighbor selection heuristics to route a packet towards the destination region. The key idea was to restrict the number of interests in directed diffusion by only considering a certain region rather than sending the interests to the whole network. Each node in GEAR keeps an estimated cost and

a learning cost of reaching the destination through its neighbors. The estimated cost was a combination of residual energy and distance to destination. The learned cost was a refinement of the estimated cost that accounted for routing around voids in the network. In case of absence of any voids, the estimated cost was equal to the learned cost. The learned cost was propagated one hop back every time a packet reaches the destination so that route setup for next packet will be adjusted.

Our work differs from GEAR in the way that the negative feedback is used. In GEAR, when a closer neighbor to the destination exists GEAR picks a next-hop node among all neighbors that are closer to the destination. If there is a hole, GEAR picks a next-hop node that minimizes some cost value of this neighbor, using the negative feedback. Unlike GEAR, in river routing we try to route smoothly around the hole in advance before hitting the obstacle.

2.2 The River Routing Algorithm

Geographic routing is becoming the protocol of choice for many sensor network applications. In the current state of the art, there are some algorithms with guaranteed delivery rate, however they require a preliminary planarization of the communication graph [BMSU99]. Planarization induces overhead and is not possible in some scenarios. On the other hand, georouting algorithms which do not rely on planarization have fairly low success rates and either fail to route messages around all but the simplest obstacles or have a high topology control overhead (e.g. contour detection algorithms) [BGJ05],[FGG$^+$05]. We present a local algorithm called *river routing* where the nodes would gradually learn about the obstacle either by hitting the obstacle, or by being informed by those neighboring nodes who hit the obstacle. The idea of river routing comes from the water current around stones (obstacles). We aim to emulate the way water flows smoothly around the stones. In the river routing algorithm when a node

reaches a local minimum, it informs its neighbors that there is an obstacle nearby, and that the obstacle is in the same direction as the destination for which the routing failed. The next time, when another node wants to send a message towards the same direction when the message reaches the area close to the obstacle, it diverts from the greedy path and goes around the obstacle. The algorithm presented here is a local algorithm and needs constant size memory in each sensor node. Our algorithm uses a weighted greedy function which takes into account the Euclidean distance to the destination as well as the probability of hitting the obstacle by moving in a certain direction. The details of the algorithm are discussed in the next section.

2.2.1 The Algorithm

In the beginning, nodes have no information about any obstacle. When a source node S wants to send a packet to a destination node D with no information available, the algorithm uses greedy routing (i.e. each node chooses the node, which is nearest to the destination amongst its neighbors as the next node). If it fails at node u (i.e. the node chosen as the next node is the same as the previous node), node u learns about the existence of an obstacle in the direction of the destination and broadcasts a message to its $k-$hop neighborhood in order to inform them that it is blocked for destination D.

Upon reception of such a message from u, the receiving node v assumes that it might be blocked for any destination within the same direction as D, and also updates its information about the distance of the closest obstacle. Each node divides its neighborhood into eight different sectors, and uses these sectors to estimate the direction as depicted in Figure 1. If node v, does not have any information about the distance for the nearest obstacle, it updates its obstacle estimate in the direction of D to be u. Otherwise, if u is closer than the current estimate, it updates its obstacle

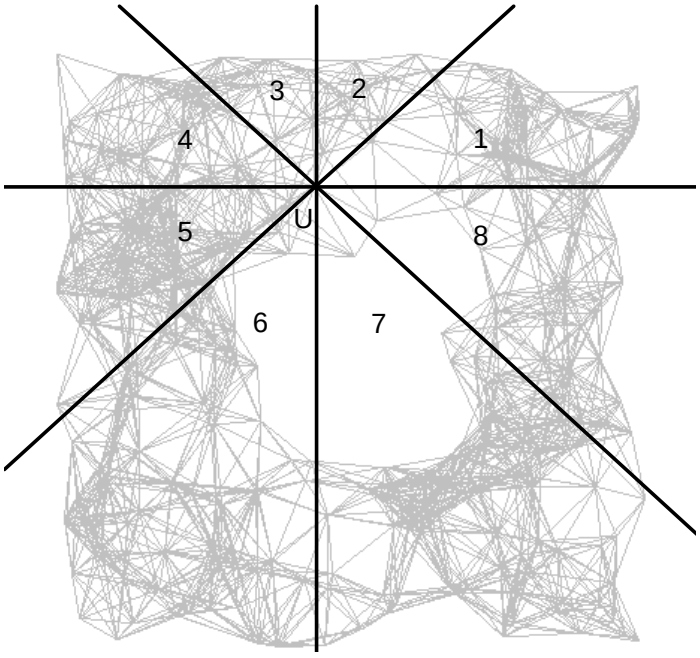


Figure 1: Dividing the graph into 8 sectors at node u.

estimate.

Furthermore, each node keeps a blocked neighborhood information list, which is a list of 3−tuples containing the sector s for which the node is blocked, the hop distance hd from the blocked node, and the number of blocked nodes for sector s at hop distance hd. When node v receives the broadcast message (u, D, k) from u, it calculates the sector s which D falls into and the hop distance hd from v. Then it verifies if a 3-tuple (s, hd, x) belongs to its blocked neighborhood information list. If such a tuple exists, the node would update its list by adding one to x (i.e. incrementing the number of blocked nodes at a certain distance for a certain sector). If not, it would just add the new tuple $(s, hd, 1)$ to its list. Afterwards, if k is not zero, node v sends the message (u, D, $k-1$) to its one-hop neighbors. Each node also keeps an array of size eight containing the farthest blocked node as an estimation of the obstacle in that direction for each sector, which is null in the beginning. Therefore, the total memory needed in each sensor node is $8(k+1)$.

Algorithm 1 River Routing Algorithm

```
procedure RiverRouting(Current, Destination)
    Next ← Null
    Previous ← Null
    while (NextNode(Previous, Current, Destination) ≠ Null)&(Next ≠
Destination) do
        Previous ← Current
        Current ← Next
        Next ← NextNode(Previous, Current, Destination)
    end while
    if Next = Null then
        Broadcast(Current, Destination, k)
    end if
end procedure
```

The river routing algorithm is described in Algorithm 1. At every step the next node is selected locally using the *NextNode* method. If no node is returned by *NextNode(previous, current, destination)*, the algorithm fails at the current node and the k-hop neighborhood of the current node are informed using the method *Broadcast(current, destination, k)*. The broadcast function takes care of updating the blocked neighborhood list as well as the obstacle estimates.

After a number of packets have been transmitted between different pairs of nodes (training phase), the nodes starts to build up information about the obstacle and their distance to the obstacle. In order to make sure that the gathered information would not cause any conflict when the source S and the destination D are at the same side of the obstacle, we would only use the weighted distance function to choose the next node only if the destination is not at the same side of the obstacle as the current node. The method *Sameside(current, destination)* in Algorithm 2 verifies if the current node and the destination are on the same side of the obstacle by verifying if the Euclidean distance between the current node and the destination node is less than the distance of the obstacle estimate in the direction of the destination for the current node, i.e. $distance(current, obstacleEstimate[sector(current, destination)]) \geq$

$distance(current, destination)$. Algorithm 2 shows how the next node is chosen in detail.

Algorithm 2 Next Node Algorithm

```
procedure NextNode(Previous, Current, Destination)
    Next ← Null
    if Sameside(Current, Destination) then
        return NextGreedyNode(Current, Destination)
    else
        for all N ∈ Neighborhood(Current) do
            if !Blocked(N, Destination)&WeightedDistance(N, Destination) <
WeightedDistance(Next, Destination) then
                Next ← N
            end if
        end for
        return Next
    end if
end procedure
```

As mentioned before, each node keeps a list of blocked nodes in its neighborhood. As the number of blocked nodes for a sector increases, it is more probable that the current node is blocked for the destination. More precisely, we consider the number of blocked nodes over the total size of that neighborhood. Clearly, a node with only 1 blocked neighbor which is its only neighbor is more probable to drop the message than a node with 10 blocked neighbors out of 100 neighbors. Thus, the ratio would give us a more precise heuristic than only the number of blocked neighbors. Besides, we would assume a stronger impact when the blocked nodes are closer to the the current node, compared to when they are farther. The weighted distance function described in Algorithm 3 takes all these into account, and it aims at balancing out the distance to the destination and the ratio of blocked neighbors for the destination direction.

Algorithm 3 Weighted Distance Algorithm

```
procedure WeightedDistance(N, Destination)
    x ← 0
    for all t = (s, k, n) ∈ blockedNeighborhoodList(N) do
        if s = sector(Current, Destination) then
            x ← x + n/neighborhoodSize(N, k)
        end if
    end for
    return EuclideanDistance(N, Destination) * (1 + x)
end procedure
```

2.3 Simulation Results

We implemented our algorithm using Java JDK 6 update 23. We model our network by a unit disk graph in an area of 200 by 200 meters, where the transmission range is 30 meters. We considered different rectangular obstacles in the middle of the graph area, with width and length equal to p percent of the width and length of the network area, respectively. We considered several obstacle sizes by varying p from 10 to 60 in increments of 10. For every obstacle size, we generated as many graphs as required so as to obtain 1000 connected graphs. In order to generate graphs with uniform distribution in the region around the void, we first generated a graph of 300 nodes with uniform distribution and then we removed any node that fell within the obstacle area. Note that although the region of the obstacle is a rectangle, using this method the void is not necessarily convex.

It should be noted that when the river routing algorithm is executed between all pairs of nodes in the graph, in many scenarios the source and destination are on the same side of the obstacle and greedy succeeds without ever hitting the obstacle. Thus, in order to show the effectiveness of our algorithm in bypassing the obstacles, in addition to calculating the delivery ratio between all pair of nodes in the graph, we also considered the success ratio of those packets, which are being transferred from one corner of the obstacle to the other corner. More precisely, nodes which are in

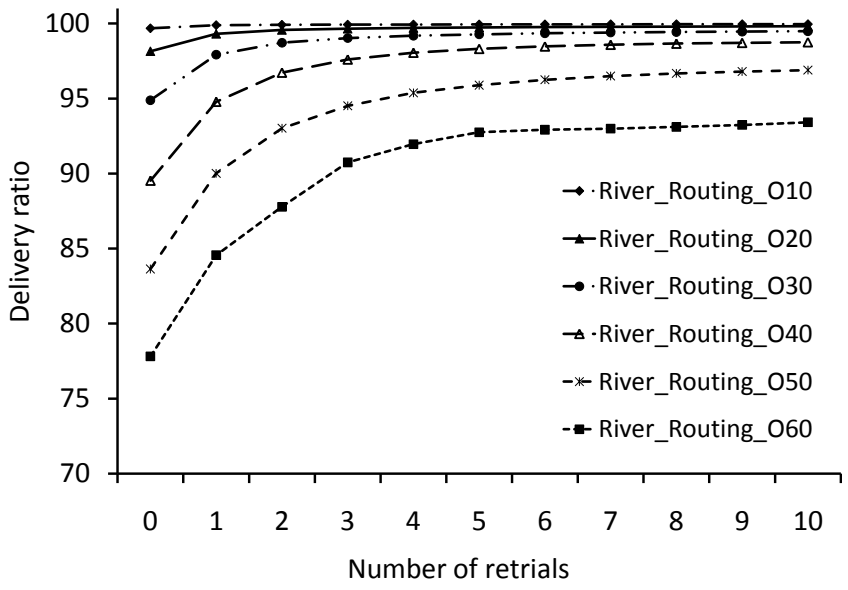


Figure 2: The effect of number of retrials on average delivery ratio

the top right (top left) corner of the network region send information to bottom left (bottom right) corner of the network region. Consequently, we present our results for the general case of routing between all pairs of nodes in the graph and corner to corner routing separately. In each case, we consider the delivery ratio as well as the average path length to study the performance of our algorithm.

2.3.1 Routing Between All Pairs

Figure 2 depicts the average delivery ratio when all nodes send a message to every other node in the network for different retrials. *River_Routing_O*p represents the river routing algorithm in a graph where the obstacle has width and length equal to p percent of the width and length of the network area, where p varies from 10 to 60 in increments of 10. It can be seen that when the number of retrials is zero, the algorithm has the same delivery ratio as the greedy algorithm [Fin87]. As the

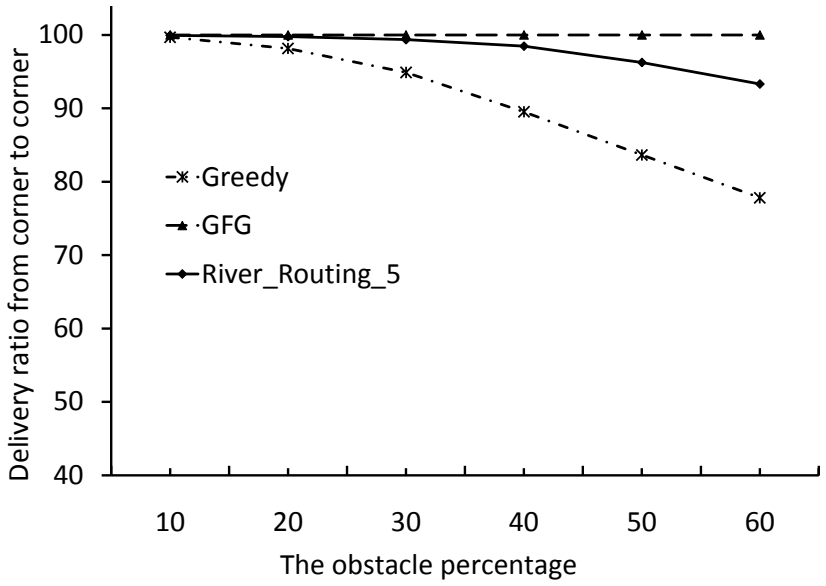
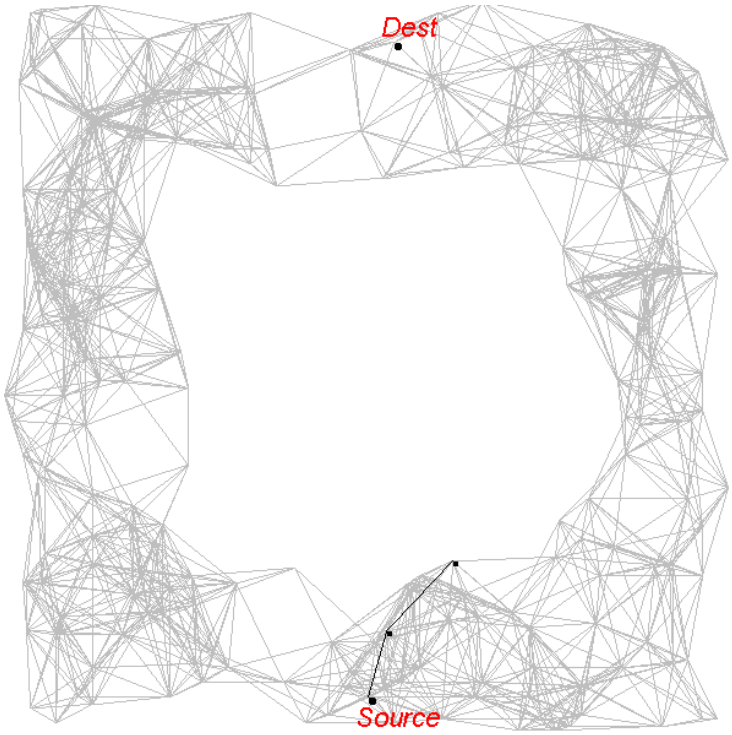


Figure 3: The effect of the obstacle size on the average delivery ratio when k is equal to 5.

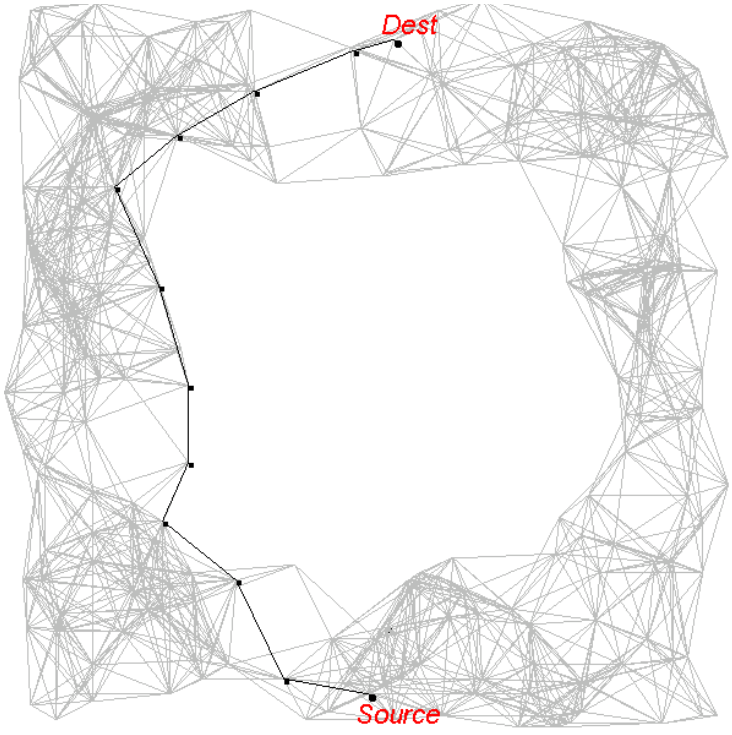
number of retrials increases, the delivery ratio increases as well. In fact, after 10 retrials, for small obstacles and even average obstacles of up to 40% the delivery ratio is at least 99%. For larger obstacles of 60%, this ratio is at most 94%. Since the gain is negligible after 5 retrials, we fix k to be 5, and we refer to it as *River_Routing_5*. We compare the delivery ratio as well as the average path length of of River_Routing_5 with the greedy [Fin87] and GFG [BMSU99] algorithms.

The average delivery ratio for River_Routing_5 is illustrated in Figure 3. As expected GFG always has a 100% delivery ratio. Although for small obstacles of 10%, both greedy and River_Routing_5 have a delivery ratio of near 100%, the difference between the delivery ratio of greedy and River_Routing_5 increases as the obstacle size increases.

Furthermore, we have examined a qualitative measure the *average path length* used by river routing from one side of the obstacle to the other side. Although the GFG algorithm in [BMSU99] always has a delivery ratio of 100% on planar graphs, it



(a) The path generated by the river routing algorithm between two arbitrary nodes u and v after the first attempt



(b) The path generated by the river routing algorithm between nodes u and v after the second attempt

Figure 4: The path generated by the river routing algorithm between nodes u and v

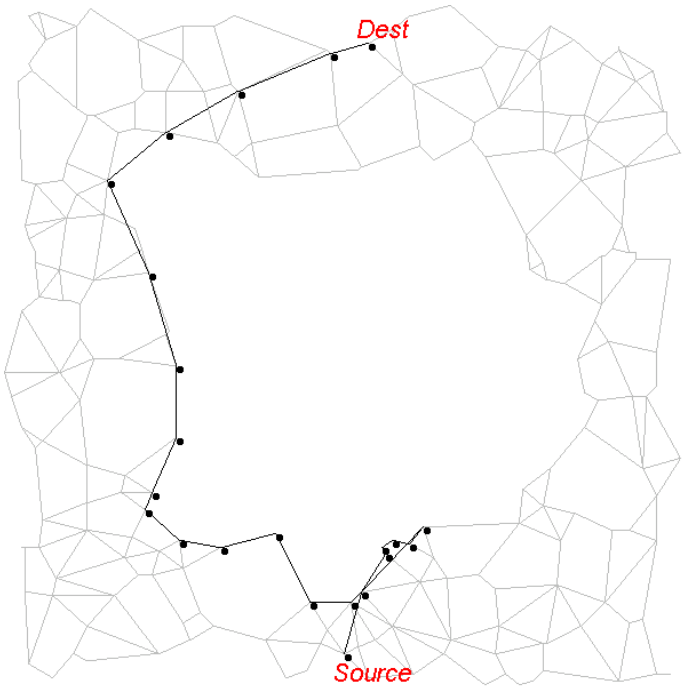


Figure 5: The path generated between the same pair of nodes u and v as in Figure 4 using GFG

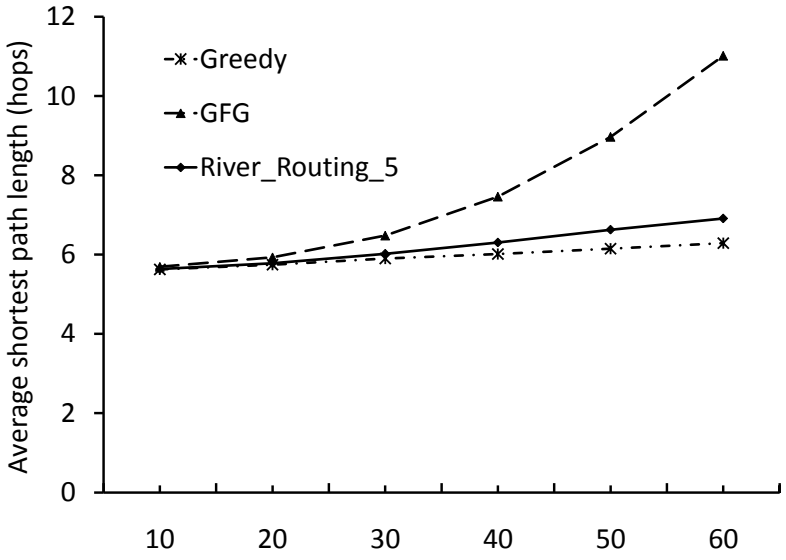


Figure 6: The effect of the obstacle size on the average path length when k is equal to 5.

doesn't necessarily choose the best path to reach the destination. Figures 4 and 5 depict different paths generated by the river routing algorithm and the GFG algorithm. It can be seen that while the path generated by river routing has length 11, the path generated by GFG between the same pair of nodes has length 24, more than twice as large as that of river routing. This difference becomes more significant in scenarios where many messages are sent between the same pair of nodes.

As illustrated in Figure 6, when the obstacle size is small, GFG has an average path length close to that of greedy. This is due to the fact that in an all to all communication pattern with a very small obstacle, the greedy part of GFG succeeds most of the time and thus there is no need for the face routing part of GFG as the recovery phase. However, as the obstacle size increases the gap between the average shortest path of greedy and River_Routing_5 widens, the River_Routing_5 manages to generate a route close to that of greedy. In fact, for a 60% obstacle, while the average path length of GFG is 75% larger than that of greedy, the average route path generated by River_Routing_5 is only 9% larger than that of greedy.

2.3.2 Corner to Corner Routing

In order to show the effectiveness of our algorithm, we study the delivery ratio as well as the average path length only between pair of nodes from one corner of the obstacle to the other corner and we refer to it as corner to corner. We first study the effect of number of retrials for different obstacle sizes. As expected as the obstacle size increases, the delivery ratio decreases. Figure 7 shows the effect of number of retrials on the delivery ratio around the obstacle for different obstacle sizes, where the neighborhood size is equal to two and maximum number of retrials is 10. When the number of retrials is equal to zero, the river routing algorithm performs as the greedy algorithm [Fin87]. Since almost all curves flatten after $k = 5$, we have fixed

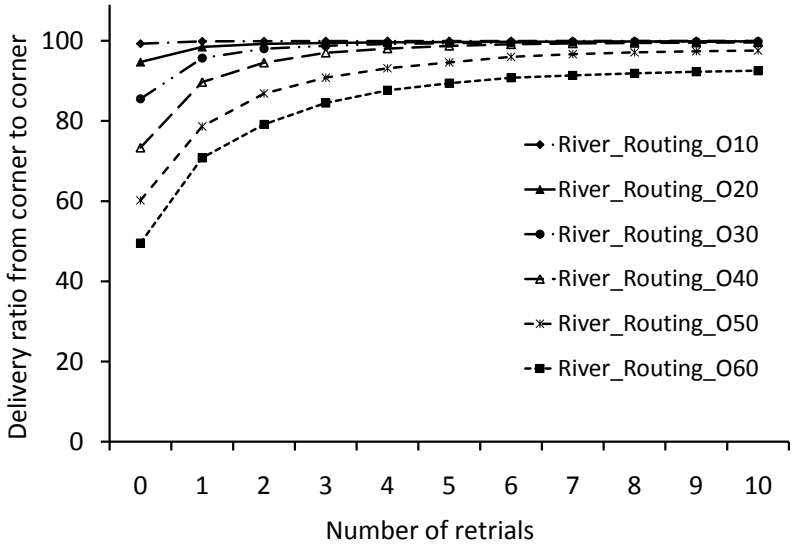


Figure 7: The effect of number of retrials on the delivery ratio from corner to corner.

the number of retrials to be 5, and we refer to the river routing algorithm with k equal to 5 as *River_Routing_5*.

Figure 8 illustrates the delivery ratio from corner to corner when k is equal to five. While, for a small obstacle of 10%, both greedy and river routing have a delivery ratio of 99%, as the obstacle increases the gap between greedy and river routing becomes bigger. In fact for a large obstacle of 60%, while greedy has a delivery ratio of 49%, after 5 retrials river routing has a delivery ratio of 90%.

We have studied the average path length parameter for different obstacle sizes and the average path length for river routing is very close to that of greedy. We have considered the path generated by the river routing algorithm after 5 trials so that the delivery ratio from corner to corner is at least 90% for all obstacle sizes and we refer to it as *River_Routing_5*. This is depicted in Figure 9. As before, for small obstacles, GFG uses its greedy component rather than the face routing component and thus it has an average shortest path length close to that of greedy. As the size

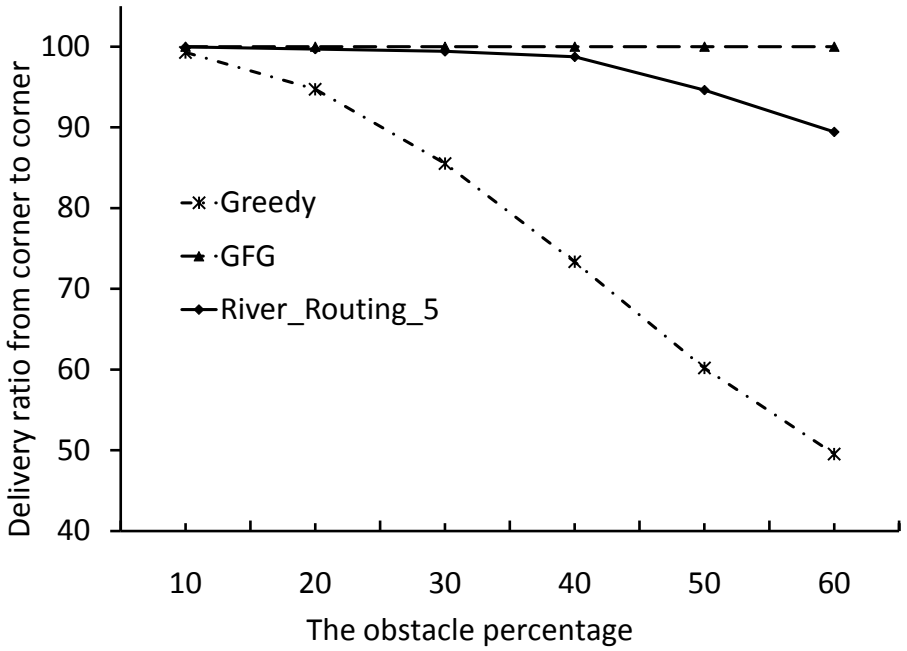


Figure 8: The effect of the obstacle size on the delivery ratio from corner to corner when k is equal to 5.

of the obstacle increases so does the gap between the average shortest path length of GFG and greedy. For a large obstacle of 60%, while River_Routing_5 has an average shortest path length of only 6% of that of greedy, the average route path generated by GFG is 58% that of greedy.

2.4 Conclusion and Future Work

We introduced a local location-aware learning routing algorithm called river routing which learns about the voids in the network using the negative feedback from other neighbors when they fail to deliver a message. Although river routing does not have guaranteed delivery, through simulations we showed that with enough number of retrials, the river routing algorithm had a high delivery ratio and it generated a path with a stretch factor very close to that of greedy. Furthermore, it should be mentioned that although there is an initial cost for retrials, they are only needed in the initial

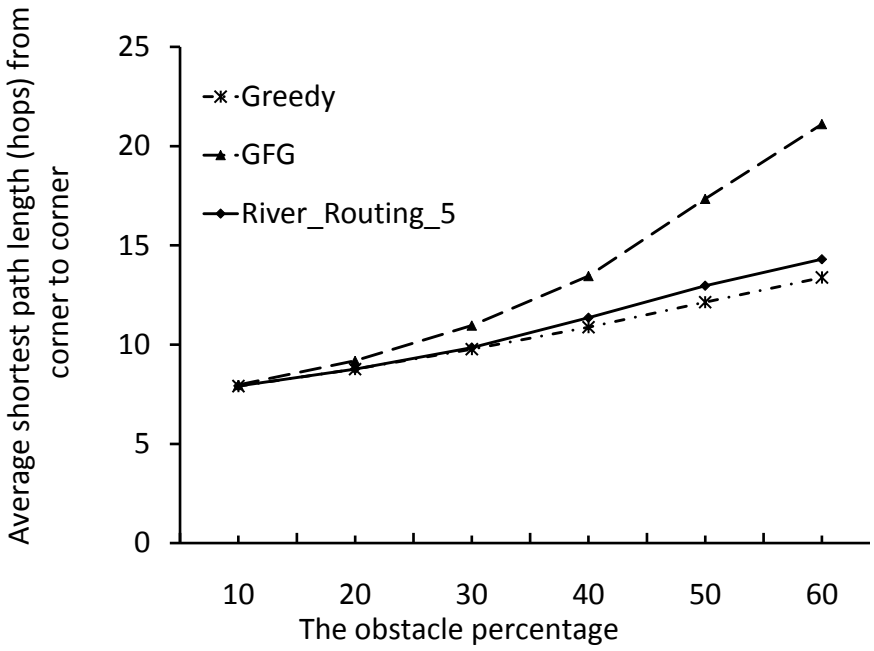


Figure 9: The average shortest path length between the all pair of nodes from the top right corner of the obstacle to the bottom left corner of the obstacle and from the top left corner of the obstacle to the bottom right corner

training phase, and once the routes are established they will be used with no needs for retrials later on during the course of the execution of the algorithm. Below we discuss some of the possible future work.

Most of the routing algorithms for ad hoc networks assume that all wireless links are bidirectional. In reality, some links may be unidirectional (different transmission ranges would need a directional model). The presence of such links can jeopardize the performance of the existing distance vector routing algorithms. One adaptation of the river routing algorithm can consider routing in directed graphs. In case the of directed graphs, the nodes informed about the obstacle might not be the same as the nodes who are trying to send the message. One can consider the problem in the class of α-reciprocal graphs, which are defined as: A directed graph $G = (V, E)$ is α-reciprocal if and only if, for every directed edge (u, v), there exist a path of length less than or equal to alpha between v and u, i.e., $(u, v) \in E \iff |P(v, u)| \leq \alpha$, where

$P(v, u)$ is the shortest path between v and u.

Furthermore, most of the routing algorithms developed for multi hop wireless networks, model the network as a two-dimensional geometric graph. However, there is increasing interest in applications where ad hoc and sensor networks may be deployed in three-dimensional space, such as in the atmosphere, or in a building. Recently, the authors in [DKN08] have shown that there is no simple local routing algorithm that guarantees delivery in unit ball graphs. They showed that for any fixed k, there could be no k-local routing algorithm that guaranteed delivery on all unit ball graphs. Furthermore, they showed that guaranteed delivery is possible if the nodes of the unit ball graph are contained in a slab of thickness $1/\sqrt{2}$. One can study the river routing algorithm by applying some restrictions on the node positions, where the network to be considered is cylindrical graph, where the difference between the z coordinates of the nodes in bounded by a constant. This can be achieved by replacing sectors with cones.

Another interesting problem to consider is the effect of river routing on congestion. Network congestion occurs when offered traffic load exceeds available capacity at any point in a network. In wireless sensor networks, congestion causes overall channel quality to degrade and loss rates to rise, leads to buffer drops and increased delays (as in wired networks), and tends to be grossly unfair toward nodes whose data has to traverse a larger number of radio hops [HJB04]. Most georouting schemes for ad hoc networks select paths using a function only based on their position and the destination position. This implicitly predefines a route for any source-destination pair of a static network, independent of the pattern of traffic demand and interference/contention among links. This may result in congestion at some region while other regions are under-utilized. However, in river routing the decisions are based on the nodes' positions as well as the information about the blocked neighborhood. This would result in

different routes each time the blocked neighborhood changes. We are expecting that the river routing algorithm helps in congestion control by selecting different routes between same pair of source and destinations at different times. Furthermore, we can modify the weighted-distance function to include congestion information in order to avoid sending messages to congested nodes.

Chapter 3

The Connected Dominating Set Problem

In this chapter, we consider the *connected dominating set* CDS problem, and we propose a distributed local approximation algorithm to find a CDS in a UDG as well as a QUDG, with constant performance ratio. We model the wireless network as a graph of location-aware nodes. Nodes could either obtain their geographical coordinates from a GPS receiver or be assigned virtual coordinates by a unique source. This information along with a tiling scheme that we will describe later helps enforce a local ordering on the execution of the algorithm. Our local algorithm is very efficient; it can be shown to have time complexity dependent only on the degree of the network. We prove that the CDS produced by our algorithm is at most a constant times the size of the optimal CDS. Additionally, simulation results on random graphs illustrate that the size of the CDS generated by our algorithm is by far smaller than other local algorithms, and even smaller than its distributed non-local competitors.

We believe that the low complexity and locality of our algorithm is as valuable as its capability in producing a very thin, tree-like CDS . The low complexity/scalability of our algorithm makes it particularly suitable for different conceivable applications

of WSNs which typically call for deployment of large and relatively dense networks. Moreover, resource constraints such as low bandwidth and limited power make the efficiency of an algorithm in terms of communication and computation a top priority. In addition, the locality of this algorithm makes it highly responsive to topological changes which are frequent in wireless networks. Topological changes might be caused as a result of node mobility, power depletion, or a node switching to sleep mode for energy conservation purposes. In such cases, it is easy to recalculate the CDS and prevent the local changes from rippling throughout the entire network.

The remainder of this chapter is organized as follows. In Section 3.1, we give an overview of the relevant previous work. The local CDS construction as well as the simulation results for UDGs is presented in detail in Section 3.2. Finally, Section 3.3 discusses different probability models for construction of the QUDGs as well as different tiling schemes, along with simulation results comparing the performance of our algorithm with several competitors.

3.1 Literature Review

Since the problem of finding a minimum dominating set is NP-hard [GJ90], a number of algorithms have been developed for the computation of a minimum dominating set (MDS) in a graph. It has been proved in [Fei98] that Chvátal's greedy algorithm's approximation ratio of $\ln \Delta$ [Chv79] is a tight bound for the computation of DS in general graphs. We will study the general trends in the formation of CDS used by the state-of-the-art algorithms and will take a look at the algorithms and heuristics that are most relevant to our proposed algorithms. These algorithms, many of which have been simulated for performance comparison in this thesis, are among the most efficient algorithms in the literature. All the studies we'll consider can be broadly categorized into two groups. The first group constructs an independent set of nodes, usually called

cluster-heads, and then connects them up by adding bridges. The second approach entails building a connected dominating set (backbone) directly without classifying nodes into cluster-heads and bridge nodes.

Although the problem of finding a minimum dominating set remains NP-hard even for the case of UDGs [CCJ90], the authors of [MBHI$^+$95] showed that constant approximation ratio is achievable. The first algorithm running in polylogarithmic-time with a non-trivial expected approximation ratio of $O(\log \Delta)$ and an approximation ratio of $O(\log n)$ with high probability was proposed by the authors of [JRS02]. Nieberg and Hurink [NH05] presented a polynomial-time approximation scheme (PTAS) for the MDS problem in UDGs. Their approach does not assume a geometric representation of the graph as the input. Given any graph as the input, their algorithm recognizes whether or not the input graph is a UDG. If so, it returns a dominating set with the approximation ratio of $1 + \epsilon$. Otherwise, it returns a certificate indicating that the input graph is not a UDG. However, since the time complexity of their algorithm is $O(n^{c^2})$ with $c = O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$, ϵ cannot be arbitrarily small in practice.

Kuhn and Wattenhofer [KW05] gave a distributed algorithm using LP relaxation techniques to compute a dominating set of expected approximation ratio of $O(k\Delta^{\frac{2}{k}} \log \Delta)$ with time complexity $O(k^2)$, where k is an arbitrary constant and Δ is the maximum node degree.

A simple local algorithm is proposed in [WL99]. This algorithm is based on a marking rule: every node with two unconnected neighbors marks itself as a dominator. This simple rule usually generates a very large CDS. To address this problem, they propose two pruning rules to reduce the size of the set returned by their algorithm. These two rules are applied to the nodes in the CDS. The first rule removes a node u if it has a neighbor v in the CDS that covers all neighbors of u. The second rule, which is an extension of the first one, eliminates a node u if it has two neighbors in

the CDS, say v and w, and all neighbors of u are dominated either by v or by w. In order to avoid simultaneous removal of neighboring nodes in the CDS, an ordering using distinct IDs is imposed. This scalable algorithm is very simple and has a low message complexity that gives it particular practical merits, but can generate a CDS that is quite large. In the worst case, it can produce a CDS as large as $\Theta(n)$ times the optimum CDS, but in practice, it demonstrates an acceptable average performance. In [DW04], Dai and Wu proposed a generalization of the two existing rules referred to as *Rule K*, in which a node u could unmark itself if it was covered by K other connected nodes. Since Rule K needed global information, they restricted Rule K to only consider immediate neighbors.

Wan et al. presented a distributed algorithm in [WAF04] which, in experiments, generated the smallest CDS size prior to our work [BMP04]. Their distributed algorithm consists of two phases. Similar to all cluster-based algorithms, first an MIS is constructed and then additional nodes are used to form a dominating tree. In constructing the MIS, they use an arbitrary rooted spanning tree T to create a ranking given by $(level, ID)$ where $level$ of a node is its hop distance, in the tree T, to the root of the tree. Constructing the tree T, which can be achieved through a distributed leader election algorithm, has a high message complexity of $O(n \log n)$ and time complexity of $O(n)$. In the second phase, a dominating tree is constructed whose internal nodes form a CDS. The time and message complexity of this phase is $O(n)$. The CDS generated by this algorithm has a constant approximation ratio of 8. In practice, it produces a very good CDS in terms of size; however, its high communication overhead and time complexity overshadow this advantage.

Basagni [Bas99] proposed an algorithm that adopts the same cluster-based approach in constructing a CDS. Their Distributed Clustering Algorithm (DCA) makes

use of a generic weight assigned to nodes to give rise to a local ordering for the execution of the algorithm. The idea is that a node decides whether to become a cluster-head or not when all its neighbors with bigger weights have made their decisions. This weight can be adjusted to select cluster-heads that have desirable properties based on a given application. For example, in order to generate a CDS of minimal size, node degree could be a good criterion for a node that intends to assume the role of a cluster-head while energy level is a more suitable metric for an algorithm whose aim is to prolong the lifetime of the network. Once an MIS is constructed using this cluster-based scheme, cluster-heads that are at most three hops apart are connected up via intermediate nodes (gateways) to form a connected backbone. DCA generates CDS's that are larger than those constructed by Wu and Li's algorithm [WL99] for relatively sparse networks and only a little better (smaller) for more dense networks. Later they added sparsification rules in [BMP04] to reduce the size of the backbone. The idea is to sparsify the CDS and generate a sparsified CDS that they call DCA-S, by breaking cycles of size 3 and 4, namely DCA-S(3), and DCA-S(4). The sparsification phase does not add much to the complexity of the algorithm, but does reduce the size of the CDS particularly as network density increases.

The first local algorithms with constant approximation ratios for both the dominating set and connected dominating set were proposed in [CDF$^+$08]. In their algorithm, they assume nodes are aware of their geographical coordinates. Using this information and a tiling scheme, they impose an ordering on the local execution of the algorithm that enforces a constant bound on the time complexity of the algorithm. The cluster-based approach employed by this algorithm results in an MIS that is at most 5 times the size of the optimal dominating set. It is further proved that the connected dominating set generated by adding bridges to the MIS has a competitive ratio of $7.453 + \epsilon$, where ϵ could be arbitrarily small.

In [WK08], a local $1+\epsilon$ PTAS for the minimum dominating and the connected dominating set problems in location aware UDGs was presented. The locality distance of their algorithm for the connected dominating set is smaller than that of [CDF$^+$08], but their dominating set algorithm has a much larger locality distance. For example, in order to achieve the same approximation ratio of 5 as in [CDF$^+$08], they use a locality distance of 917.18 times larger. Furthermore, construction of the connected dominating set is entirely dependent on the dominating set in that it uses the latter as an input. In summary, they show that, theoretically, a $1+\epsilon$ approximation ratio for the construction of DS and CDS is feasible. However, it is not a practical algorithm.

In [KMNO10], a distributed algorithm to construct a CDS has been proposed. The algorithm is different from the other studies in that instead of adding nodes to CDS, the authors proposed a coloring scheme to eliminate non-CDS nodes. At every step a node is eliminated from the set if its elimination neither leaves any node undominated or disconnects the CDS. The distributed algorithm has $O(n)$ time complexity and $O(n\Delta^k)$ message complexity, where Δ is the degree of the maximum degree node in the network and k is the hop-distance of the farthest neighbor that needs to be included in the connectivity test in the algorithm. The simulation results show that in practice the algorithm generates a thin CDS.

3.2 The CDS Problem for Unit Disk Graphs

In this section, we propose a new local algorithm to compute a connected dominating set of a location-aware UDG. To be able to make decisions locally, we need to enforce some sort of ordering that ensures the decisions made by a node only depend on the nodes within a certain distance. To achieve this, we use a tiling scheme described later. Our algorithm relies on a local spanner as a guideline for the construction of the CDS. Using this information, the nodes then select a set of candidate nodes

among themselves from which a subset of the nodes are selected as the dominating node(s) based on one of the heuristics described later in this section. When the local computation of the CDS is finished, every node in the CDS runs a local pruning test to reduce the size of the CDS.

3.2.1 Definitions and Preliminaries

We consider a wireless network of homogeneous nodes where all nodes have the same transmission range. The network is modeled by a graph $G = (V, E)$ where V denotes the set of vertices and E represents the set of edges between distinct vertices in V whose Euclidean distance is less than or equal to the transmission range of nodes. Also, throughout this study, we use the terms node and vertex interchangeably. Node u is a neighbor of node v if and only if they are adjacent in the graph. We use N_u to denote the set of neighbors of node u, referred to as neighborhood of u.

We assume that every node is aware of its geographic location. Using this information, a node determines its class number which is described in detail in the following section. A hexagon H is a logical grouping of adjacent nodes with the same class number. Connectivity between two hexagons A and B implies the existence of an edge between two nodes u and v where $u \in A$ and $v \in B$.

$N_{u,i}$ denotes those nodes in the neighborhood of u whose class number is i. $N'_{u,i}$ is the set of neighbors of u whose class number is not i; i.e. $N'_{u,i} = N_u - N_{u,i}$. H_u is the set of nodes in the same hexagon as u; i.e. $H_u = N_{u,classNum(u)} \cup \{u\}$. Finally, we say that $H_{u,i}$ is the set of nodes in the same hexagon as u, with at least one neighbor of class number i; i.e. $H_{u,i} = \{v | v \in H_u \wedge N_{v,i} \neq \emptyset\}$.

3.2.2 Tiling

We use the tiling scheme first proposed in [CDF+08] in order to ensure the locality of our algorithm. In this scheme, the plane is divided into tiles of twelve hexagons of diameter one as depicted in Figure 10. In this tiling, the first tile is placed such that the center of hexagon one is placed in coordinates $(0, 0)$. The rest of the tiles are placed so that hexagon three is adjacent with hexagons seven and ten, or hexagon eleven is adjacent with hexagons eight and four. Those edges of each hexagon that lie on the path between the top and bottom apexes in a clockwise traversal belong to the hexagon. Since every hexagon has diameter one, any two nodes within one hexagon are adjacent. A node is assigned a class number corresponding to the class number of the hexagon containing it. This approach guarantees that two nodes of the same class number are either adjacent or are of Euclidean distance greater than two, which is used to ensure the locality of our algorithm. This tiling scheme is optimal in this respect.

3.2.3 Local Spanner

As mentioned earlier, we build an approximation of a minimum spanning tree using the local algorithm proposed in [CDK+06, LWS04]. This spanner is used as a guideline to ensure that the resulting dominating set is indeed connected. In this algorithm, an edge is in the local spanner if and only if it belongs to the set of spanner edges of the graphs induced by the closed k-hop neighborhood of both its incident nodes. Our simulation results show that a very good approximation of the spanner is achievable when k equals 2 and the additional gain which is obtained by further increasing k is negligible. We only consider connecting two hexagons if there exists an edge on the local spanner between them. If there is no edge between two hexagons on the local spanner despite the fact that they are connected on the original graph, it implies that

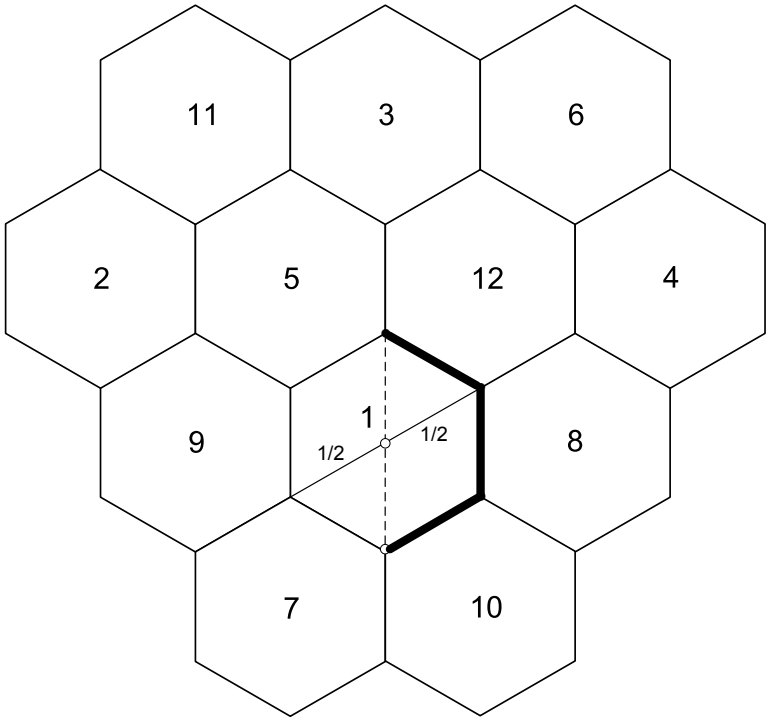


Figure 10: A tile divided into 12 hexagons of unit diameter. The bold edges belong to hexagon 1.

they are connected via some other path that goes through other hexagons.

3.2.4 Selecting the Vertices in the CDS

This section elaborates the core of our algorithm. We first describe the algorithm along with its pseudocode. Then, the different heuristics that could be used to select nodes are discussed.

3.2.4.1 Algorithm

Every node v first computes its class number using its coordinates and the tiling information. Afterwards, each node v sends a message to every neighbor u and exchanges information including the node's class number, ID and a marker value $inCDS(u)$ indicating whether the node is a dominator. If a node has the lowest

ID in its hexagon (i.e. among its neighbors with the same class number), it selects itself as the coordinator. While only coordinators execute Algorithm 4 to determine CDS nodes in their hexagons, Algorithm 5 is executed by every node in the network, whether it is a coordinator or not.

Algorithm 4 Local connected dominating set algorithm executed by coordinator u with class number cl

$H_u \leftarrow N_{u,cl} \cup \{u\}$
Wait for the lower class neighbors of H_u to finish executing the algorithm
$LCDS \leftarrow \{v | v \in H_u \wedge inCDS(v)\}$
$S \leftarrow \emptyset$
Send the message REQ_SE to all neighbors in H_u
Wait for all neighbors in H_u to send a REP_SE
When all the replies are received, S contains the class number of all hexagons i for which there exists a node in H_u with an spanner edges to H_i
if $S = \emptyset$ **then**
    $inCDS(u) \leftarrow true$
else
    for all $i \in S$ **do**
        if $\neg(\exists v \in LCDS \wedge \exists w \in N_{v,i} \wedge inCDS(w))$ **then**
            if $\exists v \in LCDS \wedge N_{v,i} \neq \emptyset$ **then**
                $dominator \leftarrow v$
            else
                Choose a node $dominator$ in $H_{u,i}$ according to the heuristic
                $LCDS \leftarrow LCDS \cup \{dominator\}$
                Send $msg = MARK\_D$ to $dominator$
            end if
            Send $msg = (SELECT\_D, N_{dominator,i})$ to $N_{dominator,i}$ via $dominator$
        end if
    end for
end if
Notify the higher class neighbors of the nodes in H_u that CDS calculation in the hexagon is finished

A coordinator node starts the algorithm only when all the lower class neighbors of the nodes in its hexagon have finished running the algorithm. When it has all the necessary information, the coordinator initializes a local connected dominating set (LCDS), which includes all those nodes in the hexagon which have been marked as

dominators as a result of dominator selection in lower class neighboring hexagons. Then it sends a REQuest Spanner Edge (REQ_SE) message to all nodes in the hexagon to determine if they have any edges on the spanner to the neighboring hexagons. All the nodes which have received the REQ_SE message, examine their neighbors and send a REPly Spanner Edge (REP_SE) message including the hexagon numbers to which they have a spanner edge to the coordinator. When the coordinator receives REP_SE messages from all its neighbors it constructs a set, which contains the class number of the neighboring hexagons which should be covered. For each class number i, two scenarios are conceivable.

(i) There already exists a node v in the local connected dominating set (LCDS) that has an edge to the neighboring hexagon H_i (i.e. $N_{v,i} \neq \emptyset$). In this case, there is no need to designate a new node to cover H_i. Furthermore, if any neighbor of v in H_i is marked as a dominator (i.e. $\exists w \in N_{v,i} \wedge inCDS(w)$), there is no need for the nodes of $N_{v,i}$ to select among themselves a new dominator as the other side of v. Otherwise, u sends a SELECT Dominator message ($SELECT\_D, N_{v,i}$) to $N_{v,i}$ via v to select a dominator among themselves according to the heuristic. Upon receiving a $SELECT\_D$ message, the receiver determines if it is the one that should be selected as the dominator among the nodes in the set according to the heuristic.

(ii) There is no node in the local dominating set that has an edge to H_i. In this scenario, any node within the hexagon with an edge to H_i is added to the candidate set. Then, the coordinator selects a node v from the set that meets the criteria of the heuristic, and sends a MARK as Dominator message ($MARK\_D$) to node v instructing v to mark itself as a dominator. In order to maintain connectivity, the coordinator node sends a message $SELECT\_D$ to v's neighbors in H_i via v to select among themselves the one that satisfies the heuristic if no such dominator already exists.

As we noted earlier, the algorithm is executed by the nodes of a hexagon only when all lower class neighbors of the nodes of the hexagon have finished their computations. Using Lemma 2 in [CDF+08], our algorithm terminates in a constant number of rounds regardless of the network size.

Algorithm 5 Local connected dominating set algorithm executed at every node u with class number cl

```
Upon receiving a message, msg:
if msg = (MARK_D) then
    inCDS(u) ← true
    Inform all your neighbors that inCDS(u) is true
else if msg = (SELECT_D, N) then
    if ¬(∃v ∈ N ∧ inCDS(v)) then
        if u is the chosen dominator from the set N then
            inCDS(u) ← true
        end if
    end if
else if msg = (REQ_SE) then
    HexSet ← ∅
    for all v ∈ N'_{u,cl} do
        if SpannerEdge(u, v) then
            HexSet ← HexSet ∪ classNumber(v)
        end if
    end for
    Send msg = (REP_SE, HexSet) to coordinator
else if msg = (REP_SE, HexSet) then
    S ← S ∪ HexSet
end if
```

3.2.4.2 Heuristics

As noted in the previous section, in the algorithm, we select a node among a set of candidate nodes in a hexagon based on some heuristic. In our experiments, we considered four different heuristics for selecting a node from the set of possible candidates. In this section, we describe these heuristics.

1. Maximum degree heuristic: In this heuristic, the node with maximum degree is selected from the set of candidate nodes. This heuristic is indeed a local

adaptation of the greedy algorithm for the set covering problem by Chvátal [Chv79]. The intuition is that the node with maximum degree is more likely to cover a larger number of nodes in neighboring hexagons. Therefore, fewer nodes need to be selected in the hexagon whose nodes are executing the algorithm in order to dominate all those neighboring hexagons.

2. Closest to the center heuristic: In this heuristic, first proposed by [CDF+08], the node that is closest to the center of hexagon is selected. The intuition is that this node would have a more symmetrical coverage of the neighboring hexagons and thus covers a larger number of hexagons compared to a node that is closer to a given neighbor and cannot cover the other neighbors. Again, this should help reduce the number of nodes that we select in the current hexagon to dominate all neighboring hexagons.

3. Longest edge heuristic: The longest edge heuristic selects the node with the longest edge to a node in the neighboring hexagon.

4. Greedy heuristic: This greedy heuristic selects the node which covers the largest number of neighboring hexagons among the candidate nodes and thus attempts to reduce the number of local dominators selected subsequently.

3.2.5 Pruning Procedure

In this section, we present our pruning procedure that can be executed locally at each node that has been selected as a dominator in the algorithm. The order in which nodes run the pruning test is similar to that of executing Algorithm 4; i.e. based on class numbers, to ensure the required consistency while maintaining the locality of the test. Furthermore, in order to make sure that the distributed execution of this test does not lead to simultaneous elimination of neighboring nodes, we assign a rank to every node. The rank of a node is an ordered pair of its class number and its ID.

A node that has been selected as a dominator waits to hear from all its lower-ranked neighbors before running the pruning procedure, which consists of evaluating two conditions. Node v meets the **domination condition** if all its neighbors have at least one other dominator. Node v meets the **connectivity condition** if the subgraph induced by its neighbors that are marked as belonging to the CDS is connected. It is clear that both these conditions can be evaluated locally by node v using information obtained from its neighbors. At this stage, node v decides to opt out of the CDS if it meets both the domination and connectivity conditions. Finally, node v informs all its neighbors about the results of its pruning procedure.

The use of a distinct rank ensures that the elimination of a node v that meets both the domination and connectivity conditions neither leaves any node un-dominated, nor disconnects the CDS.

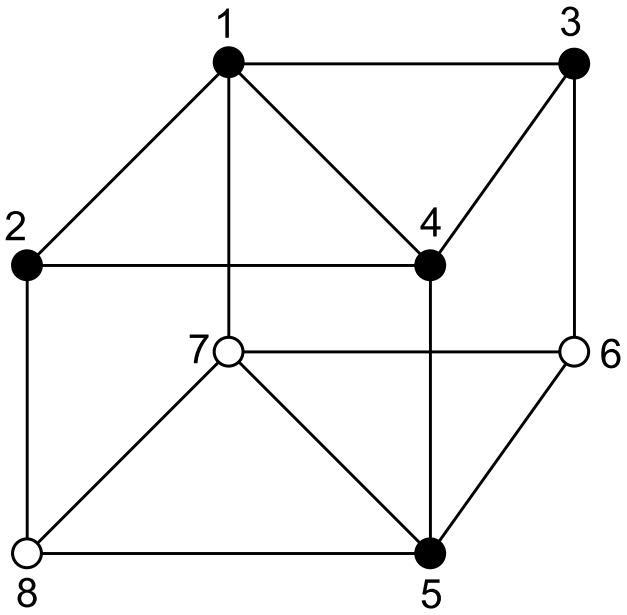


Figure 11: An example to illustrate the pruning procedure

Consider the example in Figure 11. Suppose black nodes have been selected to form the CDS in the first phase of the algorithm and white nodes are the nodes outside

the CDS (CDS = $\{1, 2, 3, 4, 5\}$). Also assume that the assignment of class numbers are such that the order in which the pruning test is run by the nodes follows their IDs. In the pruning phase, only black nodes run the pruning test to decide whether to remain in the CDS or opt out. Node 1 has four neighbors, three of which are in the CDS and therefore will not be left un-dominated if node 1 drops out of the CDS. Node 7 has also another dominator, node 5. Note that node 7's other dominator does not have to be a neighbor of node 1. Thus, all neighbors of node 1 are covered by some other node in the CDS and node 1 can proceed to connectivity test. It has three neighbors in the CDS, nodes 2, 3, and 4. These three nodes form a connected component. Therefore, node 1 is a redundant dominator and opts out. Running the same test, nodes 2 and 3 will also remove themselves from the CDS. When node 4 runs the test, the domination test fails since nodes 1,2,3 have only one dominator, node 4. Therefore, node 4 remains in the CDS. Same applies to node 5. At the end of the pruning phase, the CDS is reduced to $\{4, 5\}$. Note that nodes' IDs can affect the way pruning is carried out.

The connectivity test can be extended to check whether the set of dominators in the $k-$hop neighborhood of the dominator v form a connected component since there might be connected components of nodes in the CDS which are not detectable in the immediate neighborhood of node v. Obviously, this extension will result in pruning more nodes as k increases, but at the expense of the locality of the algorithm. Using the same tiling scheme k can have values 1 or 2, so that the pruning test is conducted in a local manner. Larger values of k requires a larger tile.

3.2.6 Proof of Correctness and Performance

We show that the set of nodes marked as dominator by our algorithm dominates or covers all nodes in the network, that it is a connected set of nodes, and finally that

it is at most a constant times larger than the optimal CDS.

Proof of coverage Since every hexagon has diameter one, it is sufficient to show that there is at least one node selected as dominator in every non-empty hexagon. We will show that this is indeed the case for the nodes marked as dominator by Algorithm 1. If all the network nodes are in the same hexagon, then the set S in Algorithm 4 would be equal to the empty set and therefore the coordinator marks itself as the dominator, which covers all the other nodes. Now, assume that the network nodes are scattered in more than one hexagon. Since the local spanner is connected, every non-empty hexagon has at least one spanner edge crossing its boundary. Therefore, the candidate set S is non-empty. If the local connected dominating set, LCDS, is non-empty, then the proof is complete. Otherwise, the algorithm states that at least one node has to be selected as the dominator to cover the neighboring hexagon H_i, where $i \in S$, which completes the proof that the set of nodes marked as dominators by Algorithm 1 dominates all nodes in the graph. We have already shown that the pruning procedure preserves the property of domination.

Proof of connectivity Assume to the contrary that the graph G_C induced by the set of nodes $C = \{v \in V | inCDS(v)\}$ produced by Algorithm 1 is disconnected. Therefore, there should be at least two separate components C_1 and C_2 in the graph. Let the set of hexagons that contain nodes in C_1 and C_2 be called HS_1 and HS_2 respectively. Note that the intersection of HS_1 and HS_2 must be empty. Also, let u and v be two nodes in C_1 and C_2 respectively. There should be a path P between u and v in the local spanner. The path P should exit HS_1 and enter another hexagon H that is not in HS_1. By our construction, HS_1 and H will be connected by some edge in G_C which implies that $H \in HS_1$, which yields the desired contradiction. This completes the proof that the set of nodes marked as dominators by Algorithm 1 induces a connected subgraph. We have already shown that the pruning procedure

preserves the connectivity property.

Proof of performance Consider the unit-radius disk centered at a node v in an optimal CDS. Clearly this disk intersects a constant number of hexagons of diameter one as used in our tiles. Since in our algorithm, in any hexagon at most eleven nodes are chosen to be in the local CDS by Algorithm 1, there are at most a constant number of nodes in our CDS (even before the pruning procedure is applied), that are contained in the disk centered at v. It should be noted that this is only a trivial upper bound before pruning is applied and a more careful analysis could, most likely, lead to a tighter upper bound. Applying the pruning procedure would considerably reduce the percentage of nodes in the CDS, but the resulting bound seems difficult to analyze. Our simulation results show that, on the average, the maximum number of nodes per hexagon before and after pruning are 2.41 and 2.04 respectively, as described in Section 4.

Thus, the following theorem is a consequence of the above discussion:

Theorem 3.2.1. *The algorithm of Section 3.4 produces a connected dominating set whose size is at most a constant times larger than the optimal CDS.*

3.2.7 Simulation Results

We compared the performance of our algorithm with several state-of-the-art algorithms through extensive simulations on randomly generated UDGs. The metrics we selected to measure in our simulations are among the most relevant factors in the study of algorithms that generate a dominating set for routing purposes ; (i) CDS size and (ii) average route length. We also investigated the impact of the locality of our pruning test on the size of the CDS. Moreover, the maximum and average number of nodes per hexagon before and after the pruning phase are presented to support the argument made earlier about the performance of the algorithm.

We used Java Platform (JDK 6 update 10) in all our simulations. Given that the transmission range of nodes and the area of the network are fixed, we vary the density of the network by assigning different values to n. In our simulations, we assigned the values 50, 100, 150, 200, 250, and 300 to n to start with a sparse network of average node degree of 3.53 and end with a dense network of average node degree of 21.2. In order to generate random UDGs with n nodes, we first created n ordered pairs of real numbers as nodes' coordinates, each generated using the Java Random class whose seed is initialized to a value based on the current system time in milliseconds. The nodes are randomly distributed in a geographic area of 200 meters by 200 meters Since the transmission range of nodes in our network model is 30 meters, two nodes are adjacent if and only if their Euclidean distance is less than or equal to 30 meters. For each value of n, we generated as many random graphs as required until we had 1000 connected graphs. The connected graphs were stored in a file and used across all simulations for the same value of n.

Figure 12 depicts a performance comparison of the four possible heuristics used in Algorithm 4. As expected, as the density of the network increases, the percentage of nodes in the CDS drops. This is due to the fact that in a dense network, the average number of nodes per hexagon increases which translates to a decrease in the ratio of the number of hexagons to nodes. As a result, the ratio of nodes in the CDS given by Algorithm 4 decreases. As shown in this figure, Max-Degree performs better than other heuristics. This result shows that Chvátal's centralized greedy algorithm [Chv79] for the set covering problem, which is known to have the best approximation ratio, might be the best in a local setting as well. While the difference is almost negligible for $n = 50$ and all four heuristics generate a CDS of size approximately 56% of nodes, it becomes more noticeable as the density grows and when $n = 300$,

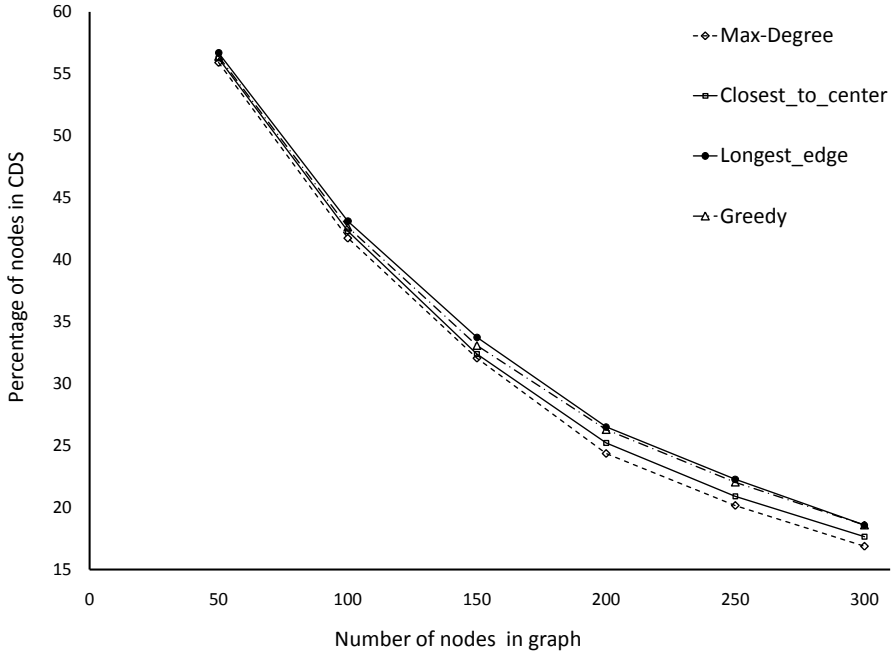


Figure 12: Percentage of nodes in the CDS for different heuristics

the difference between Max-Degree and Greedy is 10%. Our best heuristic, Max-Degree, produces a backbone consisting of only 16.8% of nodes for networks with 300 nodes.

With regard to the results obtained from the performance comparison among the four heuristics, we selected Max-Degree as our best candidate and compared it against four other most relevant algorithms presented in [CDF+08],[DW04], [WAF04], and [WL99]. The first algorithm presented in [WL99], hereafter referred to as *WuLi* after the names of the authors was selected because it is a simple and local algorithm that, like our algorithm, builds a backbone directly. The second algorithm is an extension to *WuLi* by adding a *KRestricted* rule, and therefore, we refer to it as *WuLi-KR*. The third algorithm presented in [WAF04], which we will call *WAF* after the initials of the authors' last names, is a distributed algorithm that produces the smallest

CDS size known to the authors prior to our work. Although WAF is not a local algorithm, and in this sense it is not perfectly comparable with our work, we chose it to highlight the comparative advantage of our approach in terms of backbone size. Finally, the algorithm presented in [CDF+08] which we refer to as *TBC* (tile-based and cluster-based) algorithm was selected since it is most relevant to our work in that it is a location-aware local algorithm. However, unlike our algorithm, it follows the two-step approach in CDS construction by first generating a maximal independent set (MIS) and then connecting up the nodes via bridges. We will call our algorithm *TBLS-MD* since it is a tile-based algorithm that is based on a locally constructed spanner and the Max-Degree heuristic. Note that the tiling used in our algorithm is only used to ensure locality.

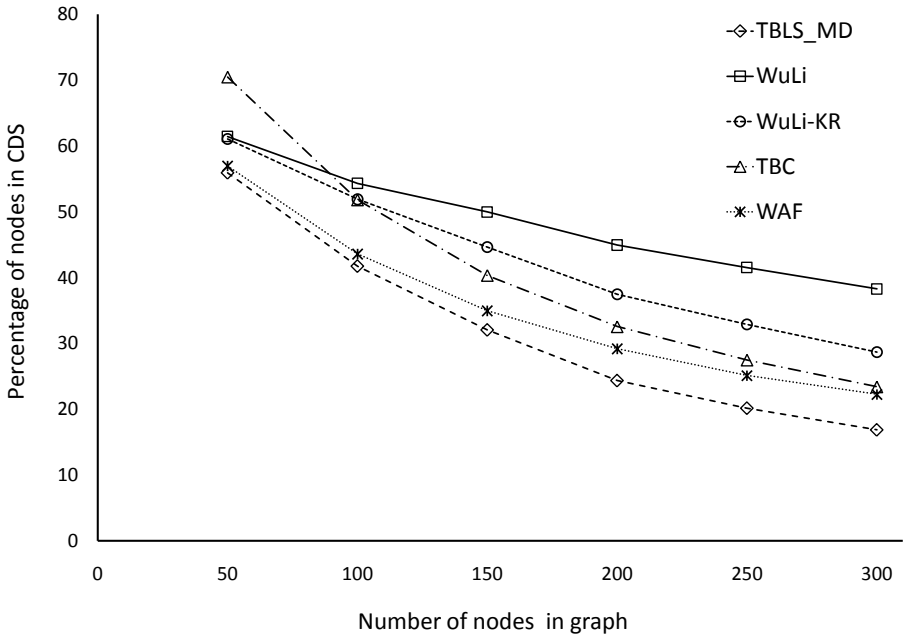


Figure 13: Percentage of nodes in the CDS for different algorithms

A comparison of the percentage of nodes in CDS with respect to network density

is illustrated in Figure 13. It can be seen that the size of the CDS generated by TBLS-MD is consistently smaller than all the other competitors. While for sparse networks ($n = 50$), TBLS-MD and WAF generate backbones of almost the same size which consist of approximately 56% of nodes, WuLi and WuLi-KR produce a backbone that is 10% larger. For such networks TBC does not perform very well mainly because it tries to push the nodes selected in MIS as far away as possible. It then has to add more nodes to bridge the gaps between these nodes which will consume a rather large percentage of the nodes. A study of other algorithms in the literature, such as DCA [BMP04], which employ the same approach shows that this method does not work well for sparse networks and generates relatively large backbones. For networks with average density ($n = 150$), TBC catches up and demonstrates a great improvement by building backbones of size 40%, whereas WuLi continues to generate a relatively large set despite an improvement of 22% in its performance due to the higher efficiency of its pruning rules for denser networks. While WuLi-KR remains the next to the last, its performance shows improvement as the density of the network increases. It generates a backbone of size 89% of WuLi, whereas it produced a backbone of almost the same size of WuLi when $n = 50$. TBLS-MD makes the most of this increase in density, both in the first phase and in the pruning phase, to generate a small backbone of 32%, smaller than the set of size 35% produced by WAF. For dense networks ($n = 300$), both TBC and WAF produce backbones of acceptably small sizes of 23% and 22% respectively, keeping in mind that TBC does so at a much lower cost due to is locality. Remarkably, TBLS-MD generates a very small backbone of size 16% in these networks, which is 25% smaller than that of WAF, and at a cost lower than it. The size of the backbone produced by WuLi-KR is still 70% larger than that of TBLS-MD and WuLi's backbone is more than twice the size of TBLS-MD's. To summarize, TBLS-MD constantly demonstrates a superior performance in terms

of backbone size and complexity among the competitors.

We also compared the average shortest path length on the backbone generated by the five algorithms TBLS-MD, TBC, WuLi, WuLi-KR, and WAF. It should be noted that the average shortest path length ratio of all our four heuristics are very close and thus we chose Max-Degree heuristic as the candidate to compare against the other algorithms. The significance of this metric lies in the essential role it plays in how well a backbone-based routing or data gathering/dissemination protocol performs. In other words, it can be deemed a major qualitative parameter in the performance evaluation of an algorithm that generates a backbone. The ratios depicted in Figure 14 were calculated as follows. Let $D(u, v)$ be the shortest path (hop distance) between nodes u and v calculated by Dijkstra algorithm [CLRS01]. The average shortest path length is $\bar{l}$, where

$$\bar{l} = \frac{\sum_{u,v \in V} D(u, v)}{\binom{|V|}{2}} \tag{1}$$

Let $D'(u, v)$ be the shortest distance between nodes u and v via the backbone. In order to calculate $D'(u, v)$, if one or both are not on the backbone, we would choose the neighbor(s) on the backbone that minimize(s) the shortest path length. The average shortest path length via backbone is then computed using Equation 1 by replacing D with D'.

As expected, since WuLi generates a rather dense backbone, the average shortest path length on its backbone is close to that of the original graph. The average shortest path on the backbone generated by WuLi-KR is at most (where $n = 300$) 5% shorter than that of TBLS-MD. Although the backbone built by TBLS-MD is smaller than both TBC and WAF, it has the same average shortest path length as TBC and a much better ratio compared to WAF. While the average shortest path length on WAF's backbone is 56% longer than on the original graph when $n = 150$,

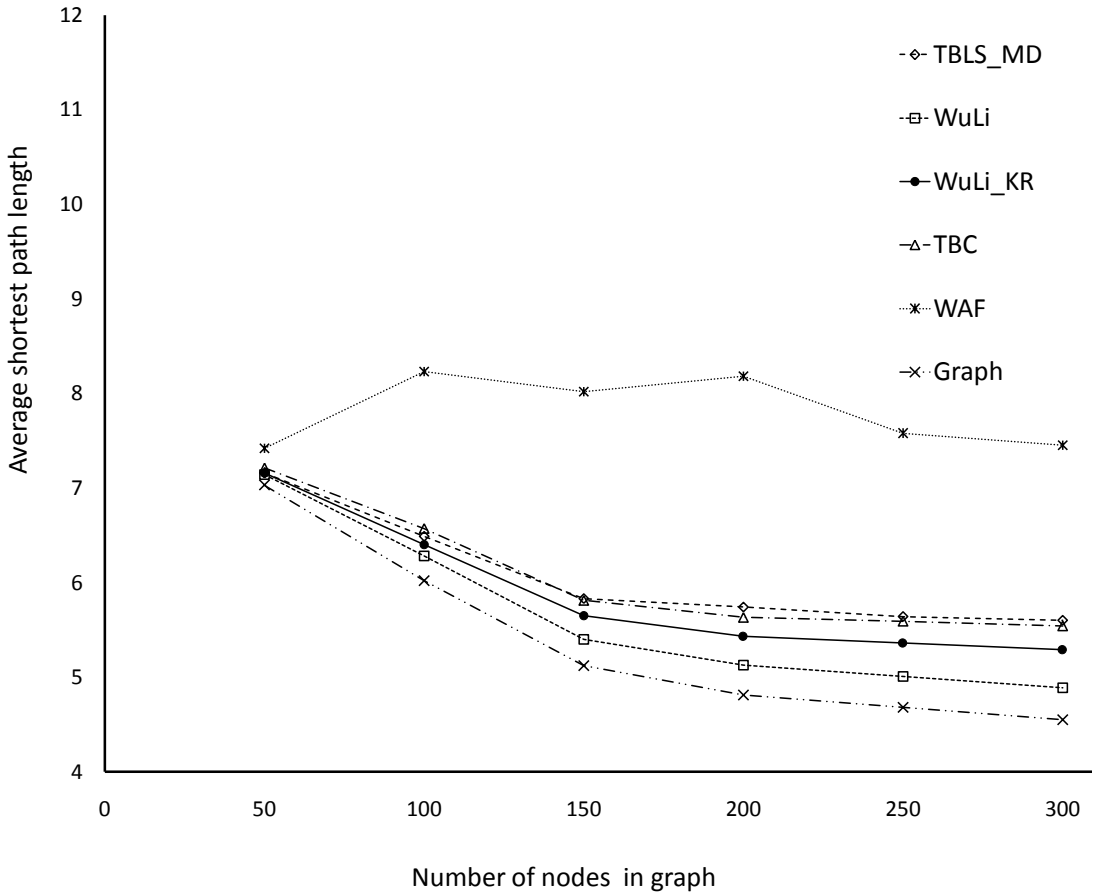


Figure 14: Average shortest path in the CDS for different algorithms

it is only 13% longer on the backbone formed by TBLS-MD. For $n = 300$, TBLS-MD continues to achieve its small ratio of 12.3%, but this ratio increases for WAF and becomes approximately 64%. In summary, both TBLS-MD and TBC generate backbones of high quality in terms of average shortest path length in spite of their small size.

In order to further study the efficiency of the pruning test and the performance of our algorithm, we conducted two other simulations which are discussed in the remainder of this section.

As mentioned earlier, our pruning test is a local test. In all the above scenarios,

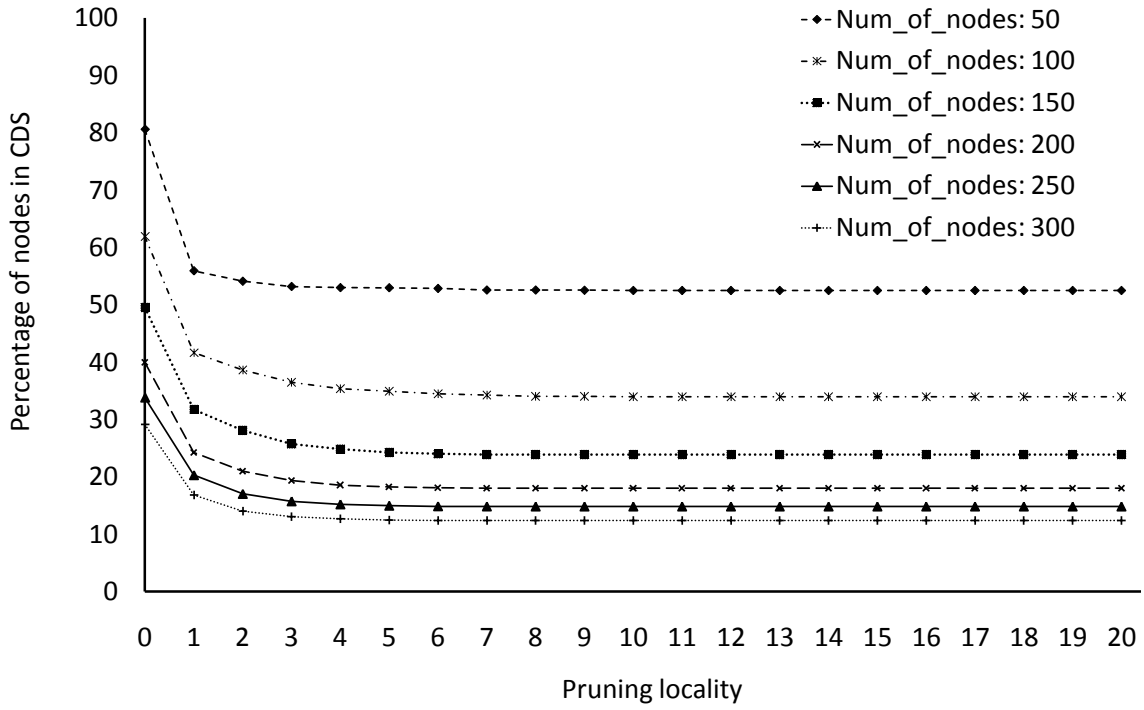


Figure 15: Percentage of nodes in the CDS for different pruning localities

we assumed that nodes perform the pruning test by just looking at their 1-hop neighborhood; i.e. ($k = 1$). However, it is conceivable that more nodes can be pruned if nodes can look farther while performing the pruning test and simulation results corroborated this intuition. In order to study the impact of the degree of locality on the efficiency of the pruning test, we increased the size of the neighborhood involved in the pruning procedure, k, from 1 to 20 and ran TBLS-MD on the same set of graphs we had generated earlier. The results depicted in Figure 15 show that it is advantageous to increase k up to four, but no considerable gain is achieved beyond that. While we achieve an improvement of 5.5% for sparse networks ($n = 50$), a relatively considerable gain of almost 33% is obtained in the case of dense networks ($n = 300$) by increasing the degree of locality to 4. It should be noted that $k = 4$ is an acceptable degree of locality. It should be mentioned that the results for values of k greater than or equal to 3 are not local, and are only presented to show that

the gain by increasing k is negligible for larger values of k. At the moment, we are investigating the tile layouts for larger values of k.

In our theoretical analysis of the algorithm, we mentioned that at most eleven nodes per hexagon could be chosen by Algorithm 1 to be in the CDS. This gives a constant upper bound on the approximation ratio of our algorithm; however, this constant is rather large. To illustrate that this is, in fact, not a very tight bound and to explain the good performance of our algorithm in practice, we measured the average and maximum number of nodes per hexagon as chosen by TBLS-MD to be in the CDS before as well as after pruning. The results are shown in Figure 16, the maximum number of CDS nodes per hexagon does not exceed 2.41 on the average and it is further reduced to 2.04 after pruning. Furthermore, the average number of nodes per hexagon does not increase in proportion with the number of nodes in the network which explains the enhanced performance of our algorithm in dense networks. As shown in Figure 16, the average number of CDS nodes per hexagon varies between 1.1 and 1.17 before pruning and fluctuates between 1.07 and 1.09 after pruning.

3.3 The CDS Problem for QUDGs

In this section, we study the performance of our algorithm in case of *quasi unit disk graphs* (QUDG), which is a more realistic model than the UDG model. In Section 3.3.1, we discuss two possible models for a QUDG. The tiling used in Section 3.2.2 cannot be used for QUDGs. We present an optimal tiling for QUDG as well as a non-optimal but efficient tiling scheme in Section 3.3.2. Finally, we compare our local algorithm with the other local CDS construction algorithms in Section 3.3.3.

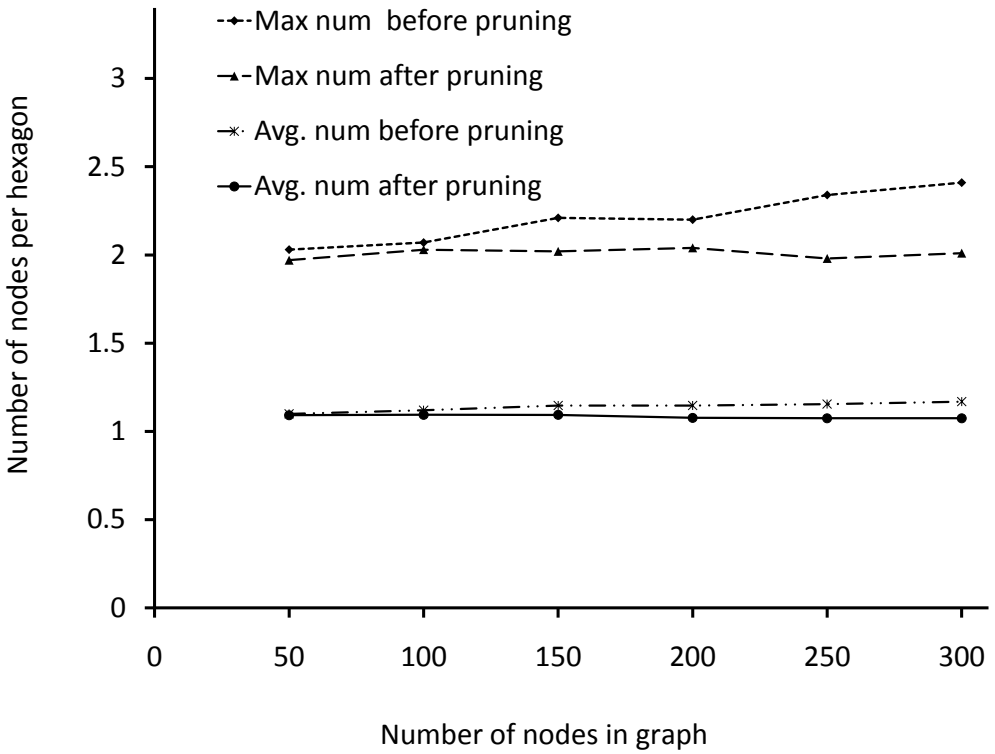


Figure 16: Number of nodes per hexagon

3.3.1 QUDG Models

In an r_QUDG, where $r \in [0, 1]$, two nodes are connected if their Euclidean distance is less than or equal to r, and not connected if their Euclidean distance is greater than one. However, if their distance is between r and 1, the two nodes may or may not be connected. We assume that there are n nodes dispersed randomly in an area $A = wl$ with maximum transmission range one. In order to model an r_QUDG, we have used two different methods to determine if an edge exists between two nodes u and v where $r < d(u, v) \leq 1$. In the first model the edge (u, v) exists with a fixed probability p, where in the second one, the closer $d(u, v)$ is to r , the more probable is the existence of the edge (u, v). In both models a node u is connected to a node v

if $d(u,v) \leq r$. We discuss the two models in the next two sections respectively.

3.3.1.1 Random Probability

In this model for a pair of nodes u and v, such that $r < d(u,v) \leq 1$, the edge (u,v) exists with fixed probability p.

$$\text{Probability(edge } (u,v) \text{ exists)} = \begin{cases} 1, & 0 < d(u,v) \leq r; \\ p, & r < d(u,v) \leq 1; \\ 0, & d(u,v) > 1. \end{cases} \tag{2}$$

Let $E(C_{u,r})$ be the expected number of nodes in circle a C in the network area A, centered at u with radius r. Thus, the expected degree of a node u is:

$$E(d_u) = E(C_{u,r}) + p * [E(C_{u,1}) - E(C_{u,r})] \tag{3}$$

$$= \frac{n\pi r^2}{wl} + p\frac{n\pi(1-r^2)}{wl}$$

$$= n\pi\frac{(1-p)r^2 + p}{wl}$$

In our simulations, we have set p to 0.5, and consequently, the expected degree is $\frac{n\pi(r^2+1)}{2wl}$.

3.3.1.2 Linear Probability

In this model, the probability that two nodes at distance between r and one are connected is proportional to their distance. More precisely, for a pair of nodes u and v, such that $r < d(u,v) \leq 1$, the edge (u,v) exists with probability:

$$\text{Probability(edge } (u,v) \text{ exists)} = \begin{cases} 1, & 0 < d(u,v) \leq r; \\ \frac{1-d(u,v)}{1-r}, & r < d(u,v) \leq 1; \\ 0, & d(u,v) > 1. \end{cases} \quad (4)$$

We now derive the expected degree of nodes using this probability model. Without loss of generality, we can assume that node u has coordinates $(0,0)$. Divide the ring between $C_{u,1}$ and $C_{u,r}$ into m rings with width equal to $\Delta = \frac{1-r}{m}$. Using the linear probability model above, if m goes to infinity, the probability of any node within ring i being connected to u is equal to $\frac{1-(r+i\Delta)}{1-r}$. Thus, the expected degree of node u is:

$$E(d_u) = E(C_{u,r}) + \lim_{m\to\infty} \sum_{i=1}^{m} [(E(C_{u,r+i\Delta}) - E(C_{u,r+(i-1)\Delta}))\frac{1-(r+i\Delta)}{1-r}] \quad (5)$$

$$= \frac{n\pi}{wl}[r^2 + \frac{1}{1-r} \lim_{m\to\infty} \sum_{i=1}^{m} [((2i-1)\Delta^2 + 2r\Delta)(1-(r+i\Delta))]]$$

$$= \frac{n\pi}{wl}[r^2 + \lim_{m\to\infty} [\frac{1}{1-r} \sum_{i=1}^{m} [(2i-1)\Delta^2 + 2r\Delta][1-r] - \frac{\Delta}{1-r} \sum_{i=1}^{m} [(2i-1)\Delta^2 + 2r\Delta]i]]$$

$$= \frac{n\pi}{wl}[r^2 + \lim_{m\to\infty} [2\Delta^2 \sum_{i=1}^{m} i - m\Delta^2 + 2rm\Delta - \frac{2\Delta^3}{1-r} \sum_{i=1}^{m} i^2 + \frac{\Delta^3 - 2r\Delta^2}{1-r} \sum_{i=1}^{m} i]]$$

Replacing Δ by $\frac{1-r}{m}$, we have:

$$= \frac{n\pi}{wl}[r^2 + \lim_{m\to\infty} [\frac{2(1-r)^2 m(m+1)}{2m^2} - \frac{m(1-r)^2}{m^2} + \frac{2rm(1-r)}{m} - \frac{2(1-r)^3 m(m+1)(2m+1)}{6m^3(1-r)}$$

$$+\frac{(1-r)^3 m(m+1)}{2m^3(1-r)} - \frac{2r(1-r)^2 m(m+1)}{2m^2(1-r)}]]$$

$$= \frac{n\pi}{wl}[r^2 + (1-r)^2 + 2r(1-r) - \frac{2}{3}(1-r)^2 - r(1-r)]$$

$$= \frac{n\pi(r^2+r+1)}{3wl}$$

3.3.2 Tiling

To ensure that all nodes in one hexagon are adjacent, we cover the area of the graph with hexagons of diameter r. Furthermore, in order for the CDS algorithm to remain local in the case of QUDGs, one should ensure that every two nodes of the same class number are either adjacent or have no common neighbor. This can be achieved by providing a tiling scheme in which every two hexagons of the same class number in different tiles have Euclidean distance greater than two. We first discuss the optimal tiling, which is only possible for certain values of r. Then, we provide a different tiling scheme, which is close to the optimal scheme, and which we have used in our simulations.

3.3.2.1 Rhombus Tiling

We start constructing the optimal tile by first calculating the minimum number of hexagons needed to ensure minimum distance two between hexagons of the same class number in different tiles. Let k be equal to the minimum number of hexagons between two non-adjacent nodes u and v of the same class number, so that the Euclidean distance between u, and v is greater than two, i.e. $k = \lceil \frac{2}{r} \rceil$.

For any pair of non-adjacent nodes u and v, if u and v have the same class number, they should be at least k hexagons apart. Thus v can not be in any hexagon less

than k hexagons away in any direction from u. The smallest parallelogram in which v cannot exist is shown in Figure 17. Thus any tile used for covering the network area should have an area at least as big as the parallelogram with sides equal to k. Note that the optimal size parallelogram is a rhombus with one of its diagonals equal to the length of its sides. The rhombus can only be constructed when k is equal to $3m + 2$, where m is some integer.

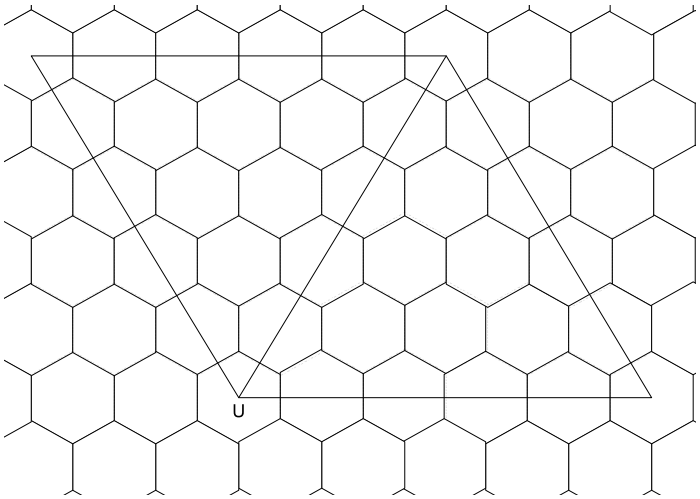
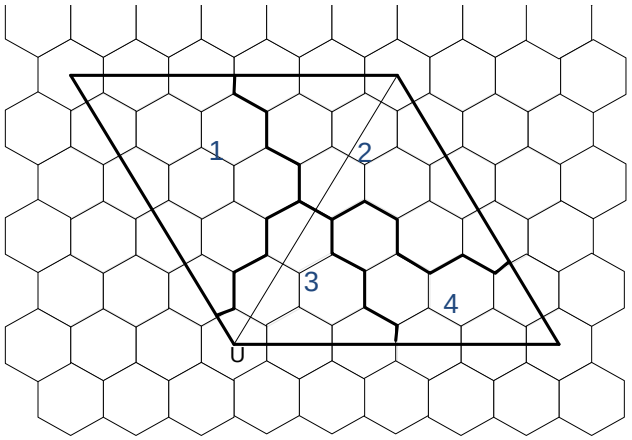


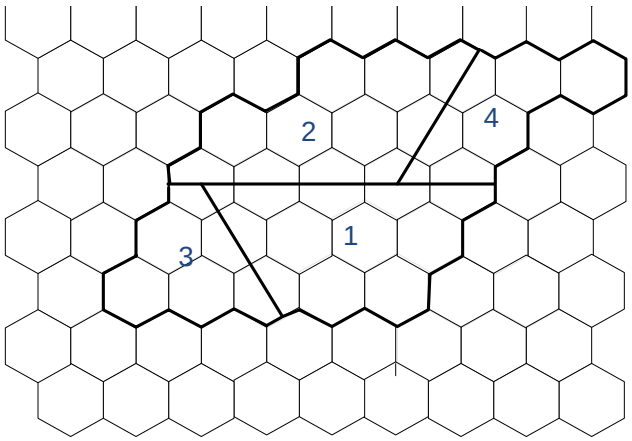
Figure 17: A Rhombus with sides equal to one of its diameters

The number of hexagons that can fit in the area of the rhombus corresponds to the minimal tile size, and the tile can be constructed by dividing the rhombus into any four parts and putting them beside each other to form a tile. One way of dividing the rhombus into four parts and the resulting tile are illustrated in Figures 18(a) and 18(b).

However, it is not always possible to construct the rhombus and the tile construction is not trivial. Furthermore, in the rhombus tile calculating the class numbers complicates the algorithm. Thus, we consider the parallelogram-shaped tiling in the next section.



(a) Dividing the rhombus into four arbitrary parts



(b) The tile constructed from the four random parts in Figure 18(a)

Figure 18: Constructing the optimal tile

3.3.2.2 Parallelogram-shaped Tiling

In this tiling scheme, we construct a k by k parallelogram-shaped tile to ensure that every two non-adjacent nodes with the same class number have distance greater than two, as illustrated in Figure 19. Thus, the minimum value of k is equal to $\lceil \frac{2}{r\sqrt{3}/2} \rceil + 1$. Clearly a tile generated using this tiling scheme has more hexagons than the optimal one. For example when $r = 1$, the optimal tile has 12 hexagons while using this tiling scheme a 4 by 4 tile is needed. However, since a parallelogram-shaped tile is close to

optimal yet easier to implement and works for all values of r, we use these tiles in our simulations. We show through simulations that even by using this tiling scheme, our algorithm outperforms the best state-of-the-art algorithms in term of the CDS size.

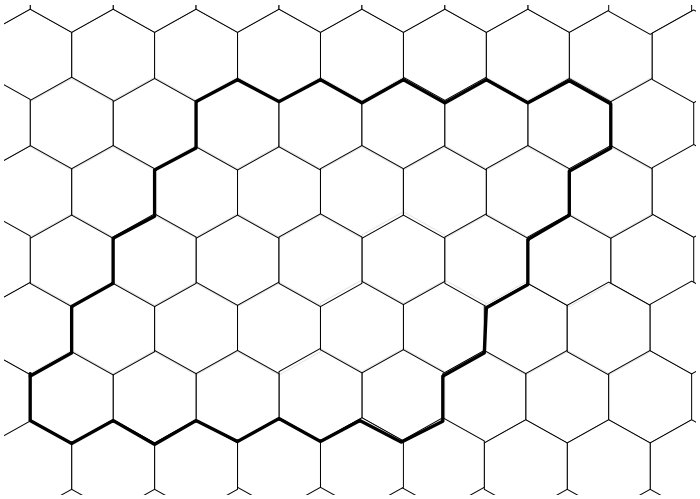


Figure 19: A 5 by 5 parallelogram-shaped tile

3.3.3 Simulation Results

We compared the performance of our algorithm with other local algorithms for construction of the CDS on randomly generated QUDGs. We used Java Platform (JDK 6 update 22) in our simulations. Given that the transmission range of nodes and the area of the network are fixed, we vary the density of the network by assigning different values to n and r, where n is the number of nodes in the network and r is the quasi factor. The nodes are randomly distributed in a geographic area of $200m$ by $200m$, with maximum transmission range 30m. In our simulations, we assigned the values 50, 100, 150, 200, 250, and 300 to n and values 0.7, 0.8, 0.9 and 1 to r to start with a sparse network of average node degree 2.58 and end with a dense network of average node degree of 21.2. In order to generate random QUDGs with n nodes, we first created n ordered pairs of real numbers as nodes' coordinates, each generated using the Java Random class and then we used both probability models mentioned

before to generate two sets of 1000 random graphs. For sparse networks, for example when $r = 0.7$ and $n = 50$ most of the random generated graphs are disconnected. We kept on generating graphs until we had a set of 1000 connected graphs for each probability model. The results for the two probability models are very close and thus here we only present the results for the linear probability model.

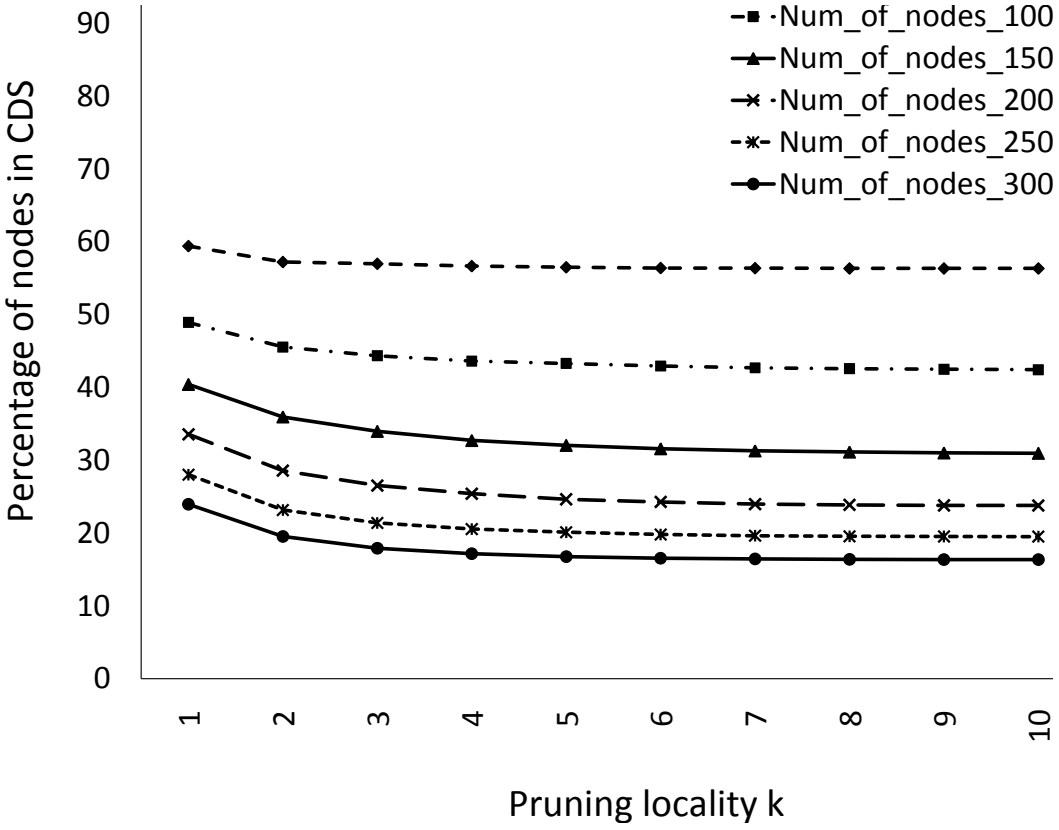


Figure 20: Percentage of nodes in the CDS for different pruning localities when $r = 0.7$

First, we executed our algorithm with pruning localities between 1 to 10 to find the trade off between the locality of the pruning procedure and the performance of our algorithm in terms of the CDS size. Figures 20 to 23 show the CDS size for different pruning localities for different quasi factors 0.7, 0.8, 0.9 and 1. For every quasi factor, we study the effect of increasing k for different node densities. It can

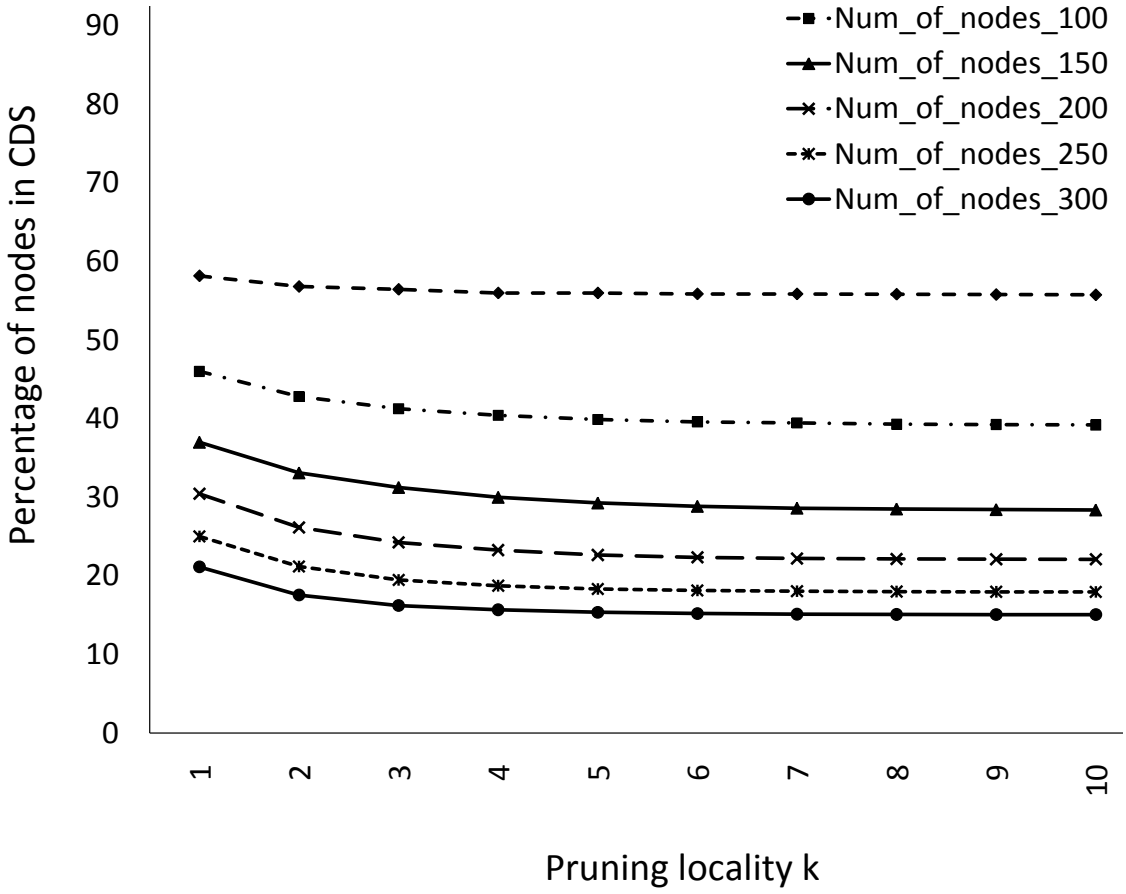


Figure 21: Percentage of nodes in the CDS for different pruning localities when $r = 0.8$

be seen in Figures 20 to 23, that for very dense networks ($n = 300$, $n = 250$) no significant improvement in the CDS size can be gained by increasing k beyond 3. It has been shown in Section 3.3.1 that the average degree of a node is proportional to the number of nodes in the network and has an inverse relation with the quasi factor. Thus, as r decreases so does the average degree of nodes, and Figure 20 shows that the CDS size can be improved by increasing k up to 5 for fewer number of nodes ($n = 50$ and $n = 100$). This has been confirmed in Figure 21 for $n = 50$ as well.Thus, we have decided to use the pruning locality $k = 4$ through our simulations, and we refer to our algorithm with pruning localities one and four as *tile-based local_1* (TBL_1) and *tile-based local_4* (TBL_4) respectively.

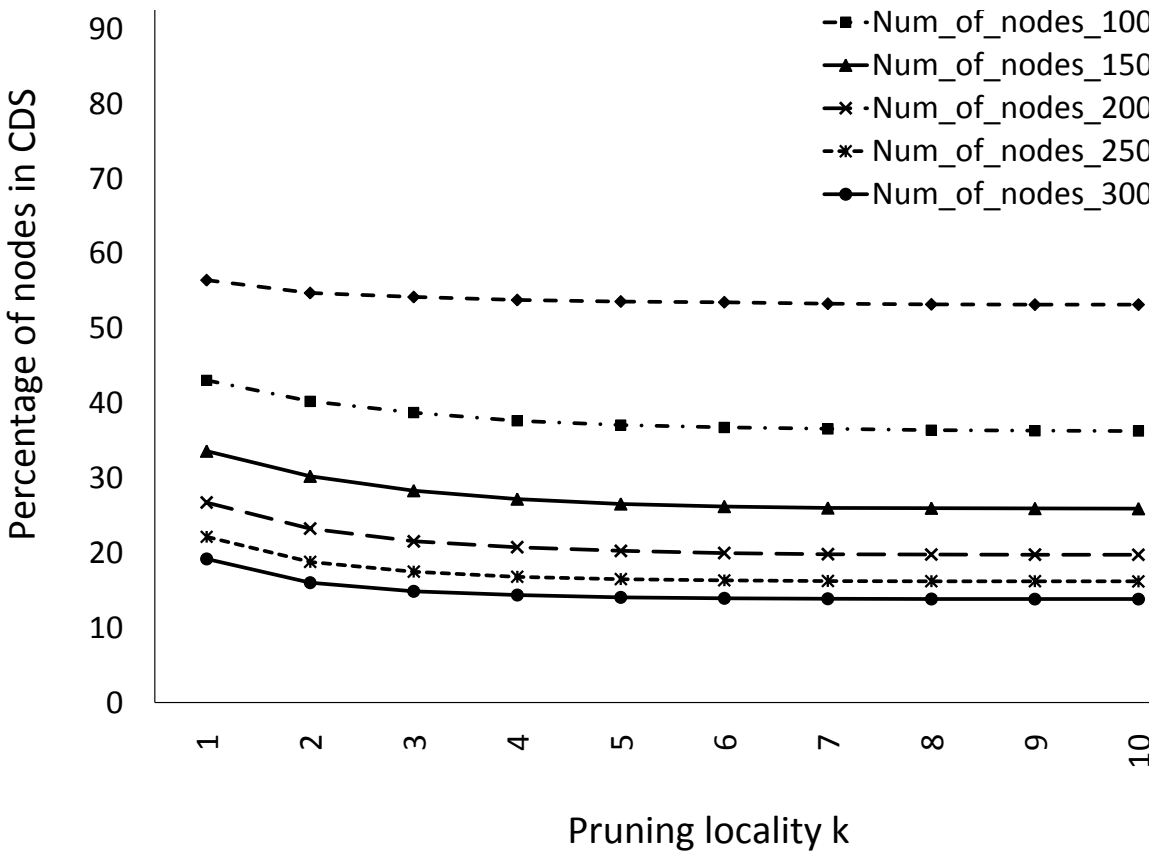


Figure 22: Percentage of nodes in the CDS for different pruning localities when $r = 0.9$

We evaluate the performance of our algorithm by comparing it with the other local algorithms for QUDGs in the literature, namely Wu and Li's local algorithm in [WL99], and the tile-based cluster-based local algorithm presented in [CDF+08]. We refer to the former as *WuLi* and the later as *TBC* throughout this section. Figure 24 to 27 depict the percentage of nodes in the CDSs produced by WuLi, TBC, TBL_1 and TBL_4 for fixed values of the quasi factor r from 0.7 to 1 with increments of 0.1. For networks with smaller expected degree, when $n = 50$, or even when $n = 100$ and $r = 0.7$, WuLi performs better than TBC. However, as soon as the number of nodes reaches 100 (150 for $r = 0.7$), TBC performs better than WuLi. In fact for very dense networks $n = 300$, the size of the CDS produced by TBC is half of that of WuLi. For both sparse and dense networks, TBL_1 and TBL_4 consistently outperform WuLi

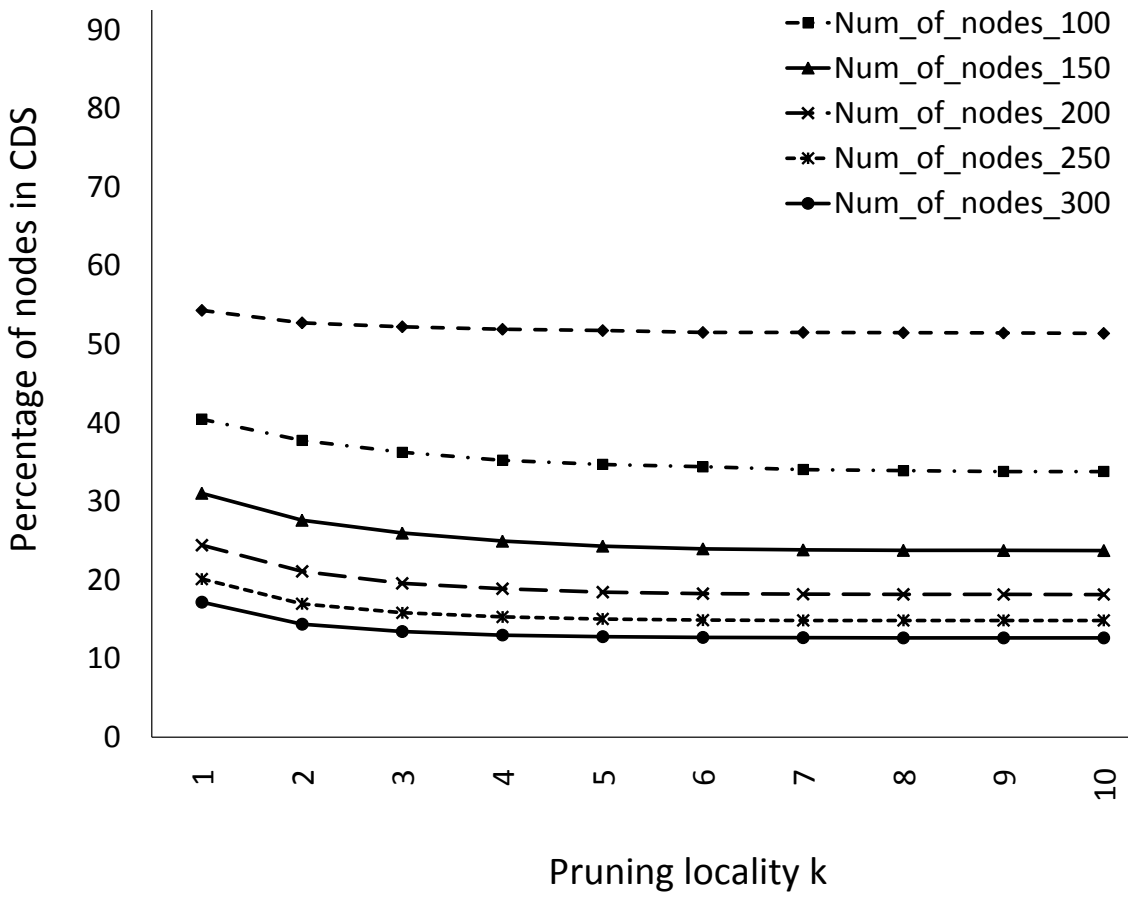


Figure 23: Percentage of nodes in the CDS for different pruning localities when $r = 1$

and TBC. Although as the density of the network increases the gap between WuLi as one group and TBC, TBL_1 and TBL_4 as another group increases, TBL still produces a thin CDS. In fact for very dense networks, the size of the CDS produced by TBC is at least 120% that of TBL_1 and 160% that of TBL_4.

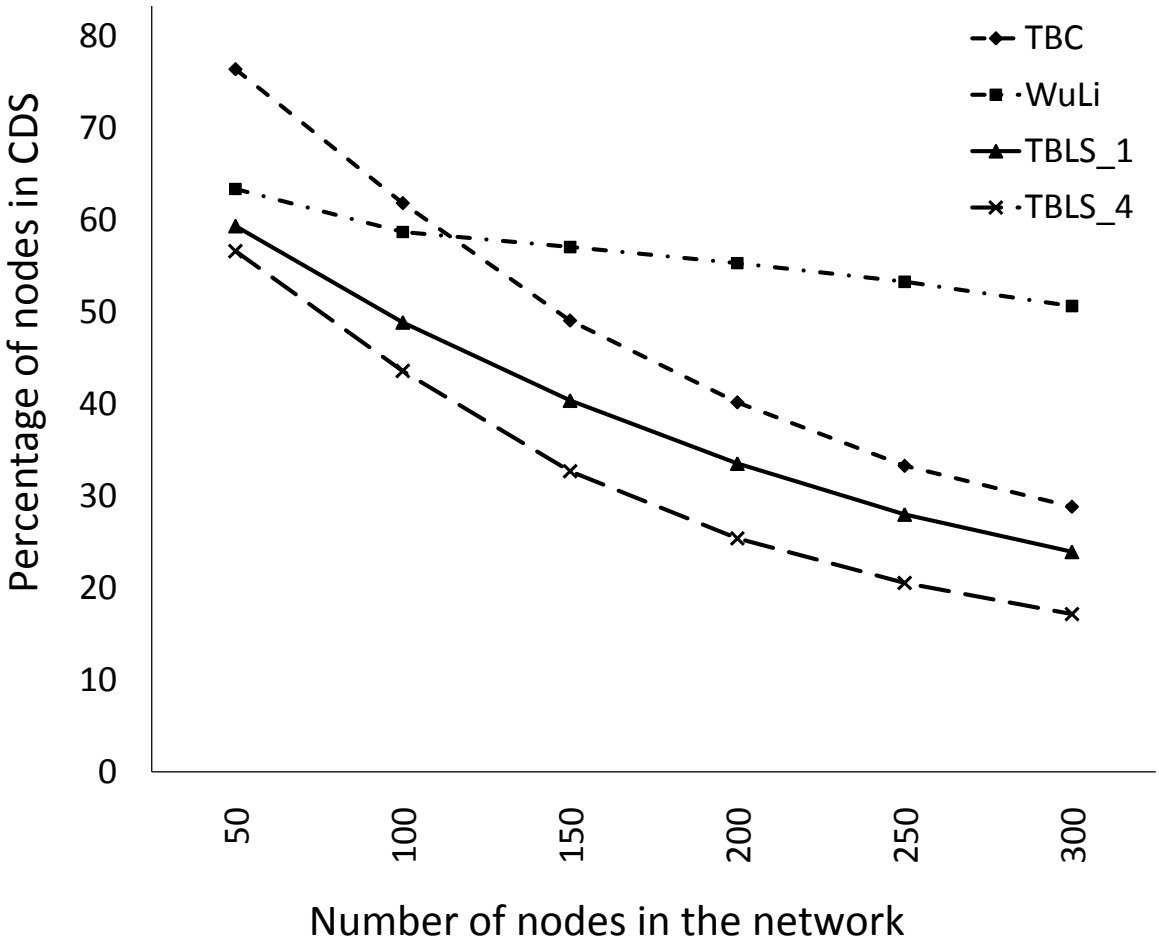


Figure 24: Percentage of nodes in the CDS for different node densities when $r = 0.7$

3.4 Conclusion

We proposed an efficient, distributed, and local algorithm to find a connected dominating set in a unit disk graph as well as a quasi unit disk graph. The CDS produced by our algorithm is provably at most a constant times larger than the optimal CDS. In our simulations on randomly generated UDGs, our algorithm produces a CDS that is significantly smaller compared to those produced by algorithms in [Bas99, CDF+08, DW04, WAF04, WL99]. It also demonstrates a good quality in terms of the average shortest path ratio.

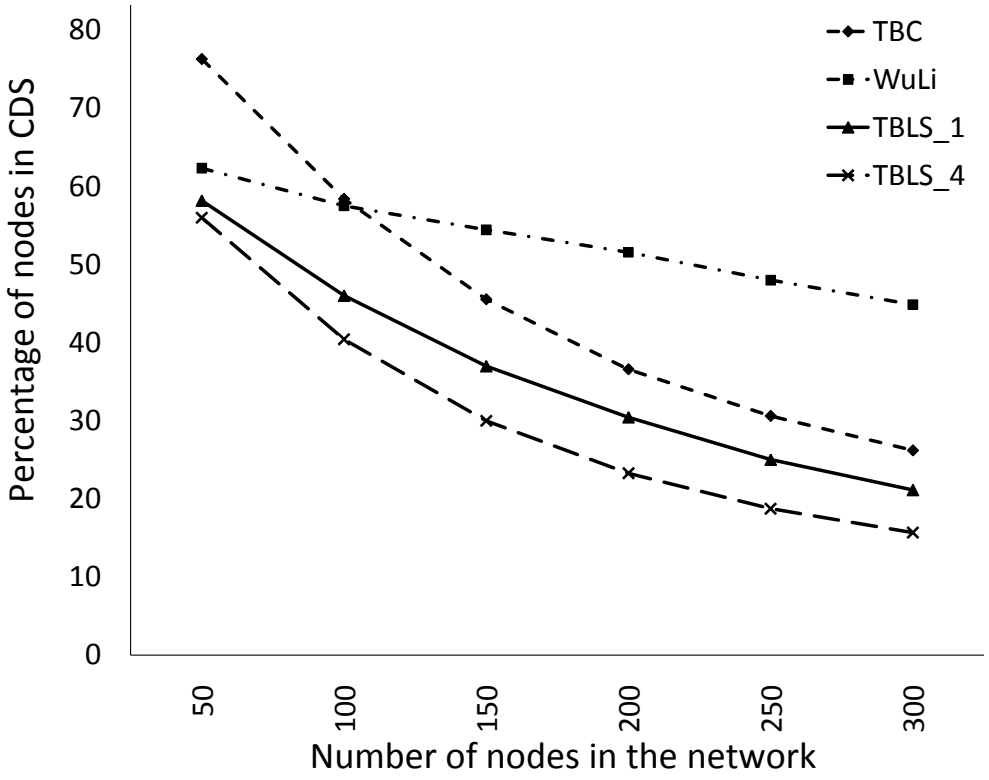


Figure 25: Percentage of nodes in the CDS for different node densities when $r = 0.8$

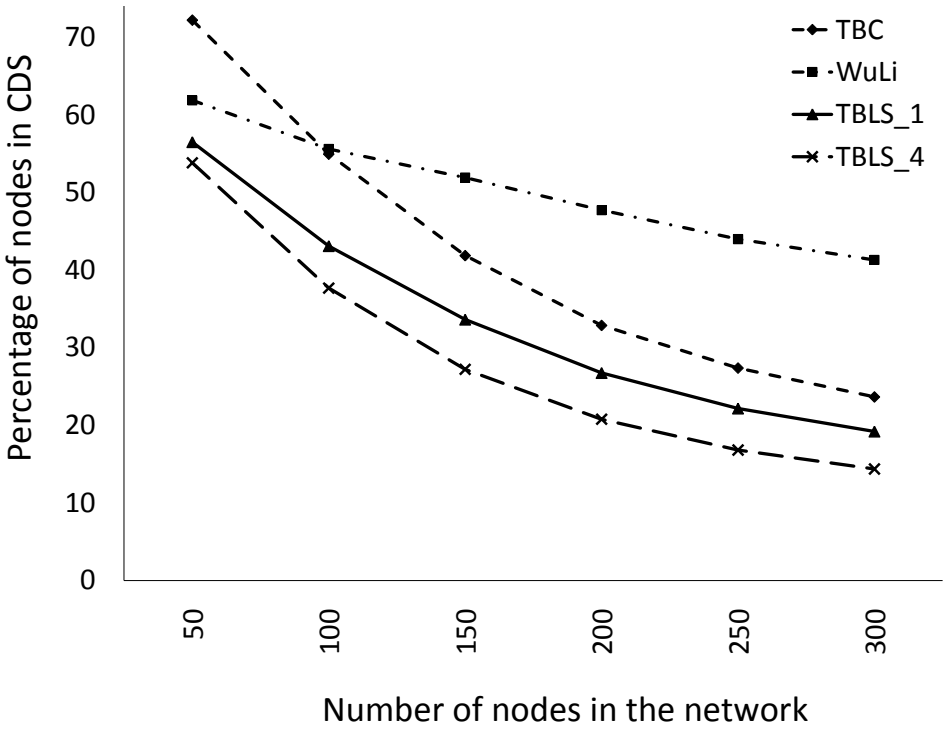


Figure 26: Percentage of nodes in the CDS for different node densities when $r = 0.9$

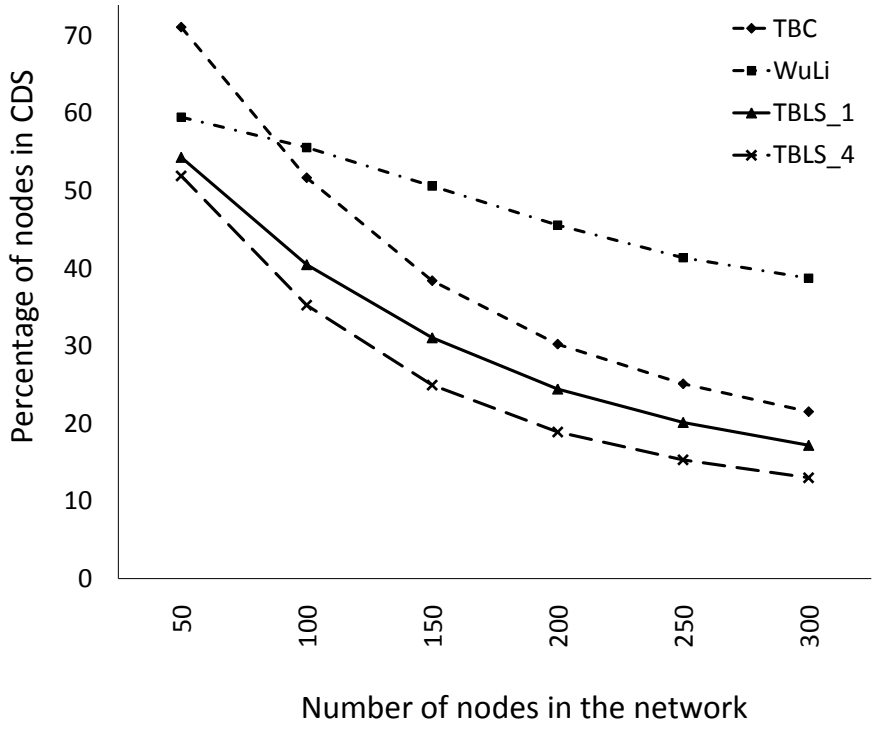


Figure 27: Percentage of nodes in the CDS for different node densities when $r = 1$

Chapter 4

Strongly Connected Dominating and Absorbent Set

In some wireless networks nodes might have different powers and transmission ranges due to different functionalities or they may adjust their transmission range for topology control purposes. Thus, some links may be unidirectional. In such cases, the network can be modeled as a *disk graph (DG)* rather than a UDG. Consequently, in this chapter we adapt our local constant ratio approximation algorithm for the construction of the MCDS of a QUDG presented in Chapter 3 to generate a strongly connected backbone in DGs. Every node in the network is either in the backbone, or it has an outgoing edge to the backbone as well as an incoming edge from the backbone. Such backbone is called a *strongly connected dominating and absorbent set* SCDAS [Wu02]. We consider the problem of finding the *minimum strongly connected dominating and absorbent set* MSCDAS.

The remainder of this chapter is organized as follows. There are only a few studies in the litreture considering the problem of construction a SCDAS in wireless networks. In Section 4.1, we discuss the few existing studies in the literature for the construction of a SCDAS in wireless networks. The local approximation algorithm

for the construction of the MSCDAS along with the definitions and preliminaries is presented in Section 4.2 followed by the simulation results comparing the performance of our algorithm with the other existing algorithms in the literature in Section 4.3.

4.1 Related Work

In [Wu02] , Wu extended the local algorithm for the construction of the CDS of an undirected graph in [WL99], to construct a local SCDAS in DGs. The algorithm used an extended marking process in which a node is marked to be included in the SCDAS, if it is on the shortest path from one neighbor to another. More formally, u changes its marker $m(u)$ to true, if there exist vertices v and w such that $(w, u) \in E$ and $(u, v) \in E$ and $(w, v) \notin E$. Furthermore, every node is assigned a unique id to give an ordering to the execution of the two pruning rules introduced to further reduce the size of the SCDAS generated by the extended marking process. The first rule removes a node u from the SCDAS if there is another node v with $id(v) > id(u)$ in the SCDAS, for which all dominating neighbors of u are also dominating neighbors of v and all absorbent neighbors of u are absorbent neighbor of v. The second rule removes a node u from the SCDAS, if there are two vertices v and w with higher ids than u , where every dominating neighbor of u is either a dominating neighbor of v or a dominating neighbor of w and every absorbent neighbor of u is either an absorbent neighbor of v or an absorbent neighbor of w. The author then discusses the implementation issues to implement the algorithm in a distributed manner. Clearly to apply either of both rules nodes require information about at least their 2-hop neighbors. However, using only 2- hop neighborhood informations enforces v and w in the second rule to be bidirectionally connected and thus this implementation version has been referred to as the *restricted implementation* and it has message complexity $O(\Delta^2)$, where Δ is the degree of the maximum degree node in the network. A more general version, not

requiring v and w to be neighbors has been introduced using 3-hop neighborhood information and is referred to as *general implementation* with message complexity $O(\Delta^3)$

The authors in [PWW+07], present centralized approximation algorithms to construct the DAS as well as the SCDAS of a DG. When the ratio of the maximum to the minimum transmission range is bounded, both algorithms have constant approximation ratios of $2.4(k+\frac{1}{2})^2 opt + 3.7(k+\frac{1}{2})^2$ and $9.6(k+\frac{1}{2})^2 opt + 14.8(k+\frac{1}{2})^2$ respectively, where k is the ratio of the maximum to the minimum transmission range. The centralized algorithm first constructs a dominating spanning tree of the outgoing edges rooted at an arbitrary node r by coloring the nodes black and blue at every other level. The algorithm then reverses the direction of all the edges, and constructs another spanning tree of the outgoing edges, rooted at the same node r. The black nodes in both trees then form a DAS and the non-leaf nodes of the spanning trees form a SCDAS. The centralized algorithm is referred to as *dominating-absorbent spanning trees DAST*. Furthermore, two heuristics for the construction of the SCDAS are presented. Both heuristics first use a subroutine referred to as *finding dominating and absorbent set* FDAS to form a DAS and then connect it up using a greedy manner. The algorithm FDAS consists of two stages, construction of a DS and construction of an AS, and returns the union of them as the DAS. The DS is generated by coloring a node u black at every step and coloring its outgoing neighbors gray until no uncolored node is remained. Node u is selected either randomly, or by choosing the highest degree node at every step. Once DS is constructed, a preprocessing is used to mark the dominated nodes (gray nodes) that are already absorbed as white nodes. Then a similar method is used to add black nodes to AS so that all gray nodes are absorbed and marks a gray node white as soon as they are absorbed by a black node. This will terminate when all gray nodes are either marked black or white. Once the DAS

calculation is completed, each of the heuristics uses a different technique to construct a SCDAS from the DAS returned by the FDAS algorithm. The first one, namely *greedy spider contraction algorithm* G-SCA, finds the largest v-spider in the graph and uses a contracting operation to form a SCDAS. A v-spider is an outconnected subtree rooted at v satisfying the following conditions: (1) all the other nodes except the root in the tree are non-white and (2) there exists a directed path in the tree from the root to every node in the tree. In the second heuristic, namely *greedy strongly connected component merging algorithm* G-GMA, two *strongly connected components* , SCCs, of black nodes are continually merged by using a shortest path between them until there is only one SCC of black nodes in the network.

In [KN10], the authors extended the distributed algorithm for the construction of the CDS in UDGs in [KMNO10], to construct the SCDAS in DGs. They first presented a centralized description of their algorithm in which all nodes in the network are initially in the SCDAS. Every node is assigned a unique rank which is an ordered pair of its number of neighbors in SCDAS and its id. At every iteration of the algorithm a node u with the minimum rank runs a *dominating and absorbent test* DAT to verify if all its outgoing neighbors have a dominator other than u, and if all its incoming neighbors have an absorbent neighbor other than u. If the DAT is successful, a connectivity test is executed which verifies if the subgraph induced by neighbors of u in SCDAS form a strongly connected component. A node u removes itself from the SCDAS if both tests are successful. Furthermore, a distributed implementation of the algorithm, namely *PInOut_UD* is presented in which initially all nodes have a *pending status*. Every node u exchanges its rank with all its neighbors and can only run the algorithm if none of its lower ranked neighbors have a pending status. Once a node finishes running the algorithm it will change its status to either *in* or *out* and will send a message to all its neighbors along with its new status. The authors

also discuss the k-hop extension of the connectivity test and refer to the algorithm as *PInOut_UDk*, for different values of k. The *PInOut_UDk* has time complexity $O(n^2)$ and message complexity $O(n^2k^2)$. Through simulations, they showed that two variants of their algorithm, namely PInout_UD1 and PInout_UD4 produce a thin SCDAS and outperform the algorithms presented in [PWW+07] and [Wu02].

4.2 The Local Approximation Algorithm for the Construction of the MSCDAS

Before presenting the algorithm, we first present some of the terms and definitions used in this section.

4.2.1 Definitions and Preliminaries

We model the network with asymmetrical links with a disk graph $G = (V, E)$, where V is the set of vertices and E is the set of directed edge. Every node i in V has a range r_i in $[r_{min}, r_{max}]$. For every vertex j, there is a directed edge between vertices i and j if and only if the Euclidean distance between i and j is less than or equal to r_i, $(i, j) \in E \iff dist(i, j) \leq r_i$.

In a directed graph, we say node u is dominated by node v, if there exists an incoming edge (v, u) from v to u. Analogously, u is absorbed by node v, if there exists an outgoing edge (u, v) from u to v. A set $A \subseteq S$, is a *dominating and absorbent set* DAS of S, if for every vertex $u \in S - A$, there are nodes v and w in A (not necessarily distinct), where u is dominated by v and is absorbed by w, $(v, u) \in E$ and $(u, w) \in E$. If the vertices in A induce a strongly connected graph, A is called a *strongly connected dominating and absorbent set* SCDAS. We use $N_d(u)$ to denote the dominating neighbor set of node u, i.e. $N_d(u) = \{v|(v, u) \in E\}$. A node

$v \in N_d(u)$ is also referred to as an *incoming* or *ingress* neighbor of node u in the literature. Likewise, $N_a(u)$ is used to denote the absorbent neighbor set of node u, i.e $N_a(u) = \{v|(u,v) \in E\}$. A node $v \in N_a(u)$ is also referred to as an *outgoing* or *egress* neighbor of u in the literature. Every node u has a rank $(classNumber(u), id(u))$ which is an ordered pair of its class number and *id*, where the class number indicates the number of the hexagon that contains u, using the parallelogram-shaped tiling scheme presented in Chapter 3. Assigning a unique *id* to every node ensures that when comparing nodes' ranks ties are broken.

4.2.2 The Algorithm

Every node first computes its class number using its coordinates and the tiling information, where the diameter of the hexagons is equal to the minimum range in the network. This ensures that all nodes in the same hexagon are connected via bidirectional links, and as long as at least one node per hexagon is selected the resulting set is a DAS. Each node sends a message to its neighbors and exchanges information including the node's class number, id and a marker value indicating whether the node is in the DAS. Once all nodes in the hexagon have exchanged messages with their neighbors in the other hexagons, every node u sends $N_d(u)$ and $N_a(u)$ to the cell coordinator which is lowest id node in the hexagon (i.e. among its neighbors with the same class number). While only coordinators execute Algorithm 6 to determine SC-DAS nodes in their hexagons, Algorithm 8 is executed by every node in the network, whether it is a coordinator or not.

The coordinator starts the algorithm only when all the lower class neighbors of the nodes in its hexagon have finished running the algorithm. All nodes v in the hexagon send the set of their dominating neighbors $N_d(v)$ as well as their absorbent neighbors $N_a(v)$ to the coordinator. The coordinator then uses this information to

Algorithm 6 Local strongly connected dominating and absorbent set algorithm executed by coordinator u with class number cl

```
H_u ← N_{u,cl} ∪ {u}
Wait for the lower class neighbors of H_u to finish executing the algorithm
Wait for all neighbors v in H_u to send a N_d(v) and N_a(v)
S_d ← ∅
S_a ← ∅
for all v ∈ H_u do
    for all w ∈ N_d(v) do
        if classNumber(w) ≠ cl ∧ classNumber(w) ∉ S_d then
            S_d ← S_d ∪ {classNumber(w)}
        end if
    end for
    for all w ∈ N_a(v) do
        if classNumber(w) ≠ cl ∧ classNumber(w) ∉ S_a then
            S_a ← S_a ∪ {classNumber(w)}
        end if
    end for
end for
if S_a ∪ S_d = ∅ then
    inDAS(u) ← true
else
    ConnectUTo(S_a, S_d, H_u)
end if
```

determine the set of class numbers S_d with outgoing edges to nodes in H_u and the set of class numbers S_a with incoming edges from H_u. For every class number i in S_d (S_a), Algorithm 6 first verifies if there is already a node in H_u that has been selected as a dominator or absorbent node with an incoming (outgoing) edge from (to) i. If such node does not exist, a node in H_u is selected as a dominator (dominatee) according to one of the heuristics mentioned in Chapter 3, and a $SELECT\_DAS$ message would then be sent to all neighbors of dominator (dominatee) in i, so that at the end of the algorithm there is an incoming (outgoing) edge in the DAS from (to) i. This ensures the connectivity of the backbone.

Once Algorithm 6 has terminated, we use an adaptation of the pruning procedure presented in Chapter 3 , to further reduce the number of nodes in the SCDAS. A node

Algorithm 7 *ConnectUTo*(S_a, S_d, H_u): Choose local DAS nodes in H_u such that the resulting DAS is connected to DAS nodes in neighboring hexagons with class numbers in S_a and S_d

for all $i \in S_d$ **do**
  if $\neg(\exists v \in H_u \wedge inDAS(v) \wedge \exists w \in N_d(v,i) \wedge inDAS(w))$ **then**
    if $\exists v \in v \in H_u \wedge inDAS(v) \wedge N_d(v,i) \neq \emptyset$ **then**
      $dominator \leftarrow v$
    else
      $C_d \leftarrow \{v \in H_u \wedge inDAS(v) \wedge N_d(v,i) \neq \emptyset$
      Choose a node *dominator* in C_d according to the heuristic
    end if
    Send $Mark\_DAS$ to *dominator*
    Send $SELECT\_DAS(N_d(dominator, i))$ to $N_d(dominator, i)$ via *dominator*
  end if
  if $\neg(\exists v \in H_u \wedge inDAS(v) \wedge \exists w \in N_a(v,i) \wedge inDAS(w))$ **then**
    if $\exists v \in v \in H_u \wedge inDAS(v) \wedge N_a(v,i) \neq \emptyset$ **then**
      $dominatee \leftarrow v$
    else
      $C_a \leftarrow \{v \in H_u \wedge inDAS(v) \wedge N_a(v,i) \neq \emptyset$
      Choose a node *dominatee* in C_a according to the heuristic
    end if
    Send $Mark\_DAS$ to *dominatee*
    Send $SELECT\_DAS(N_a(dominatee, i))$ to $N_a(dominatee, i)$ via *dominatee*
  end if
end for
Notify the higher class neighbors of the nodes in H_u that CDS calculation in the hexagon is finished

that has been selected as dominator or dominatee waits to hear from all its lower-ranked neighbors before running the pruning procedure, which consists of evaluating two conditions. Node v meets the **domination an absorption condition** if all its neighbors have at least one other dominator and one other absorbent neighbor. Node v meets the **strong k-connectivity condition** if the subgraph induced by its k-hop neighbors that are marked as belonging to the SCDAS is strongly connected. It is clear that both these conditions can be evaluated locally by node v using information obtained from its k-hop neighbors. At this stage, node v decides to opt out of the SCDAS if it meets both the domination and strong k- connectivity conditions. Finally,

Algorithm 8 Local strongly connected dominating and absorbent set algorithm executed at every node u with class number cl

```
Upon receiving a message, msg:
if msg = (MARK_DAS) then
    inDAS(u) ← true
    Inform all your neighbors that inDAS(u) is true
else if msg = SELECT_DAS(N) then
    if ¬(∃v ∈ N ∧ inDAS(v)) then
        Choose a node u in N according to the heuristic
        inDAS(u) ← true
    end if
end if
```

node v informs all its neighbors about the results of its pruning procedure. Like before, the use of a distinct rank ensures that the elimination of a node v that meets both the domination and strong connectivity conditions neither leaves any node un-dominated or un-absorbed, nor disconnects the SCDAS.

4.3 Simulation Results

To evaluate the performance of our algorithm in networks with unidirectional links, we conducted a series of simulations to study the impact of different node densities, different pruning localities and the different percentage of unidirectional links on the size of the constructed SCDAS.

Extensive simulations to study the performance of all the few algorithms presented for the construction of a SCDAS in directed graphs prior to our study has been conducted in [KN10]. In our simulations, we have used the same series of graphs used in [KN10] and we reproduced the results. In all input graphs, nodes are randomly distributed in a geographic area of $200m$ by $200m$, and the density is varied by changing the number of nodes from 50 to 300 in increments of 50. Furthermore, to study the effect of percentage of unidirectional links on the size of the SCDAS, random

transmission ranges in $[r_{min}, r_{max}]$ are assigned to every node, where r_{max} is $50m$ and r_{min} has values $10m$, $20m$, $30m$, $40m$ and $50m$ so as to simulate different percentage of unidirectional links in the networks. For each value of (n,r_{min}), as many random graphs as required are generated until there are 1000 strongly connected graphs. The graphs were stored in files and used across different simulations using different algorithms. As before, we used Java Platform (JDK 6 update 10) for our simulations.

First, we investigated the effect of the locality of the strong k-connectivity test of the pruning procedure to determine the best trade-off between the degree of locality in this test and the number of nodes that can be pruned. It can be seen that in sparse graphs $n = 50$ and $n = 100$, specially when the average number of neighbors with bidirectional links is small we can not benefit from increasing k by more than 5. This is illustrated in Figures 28 and 29.

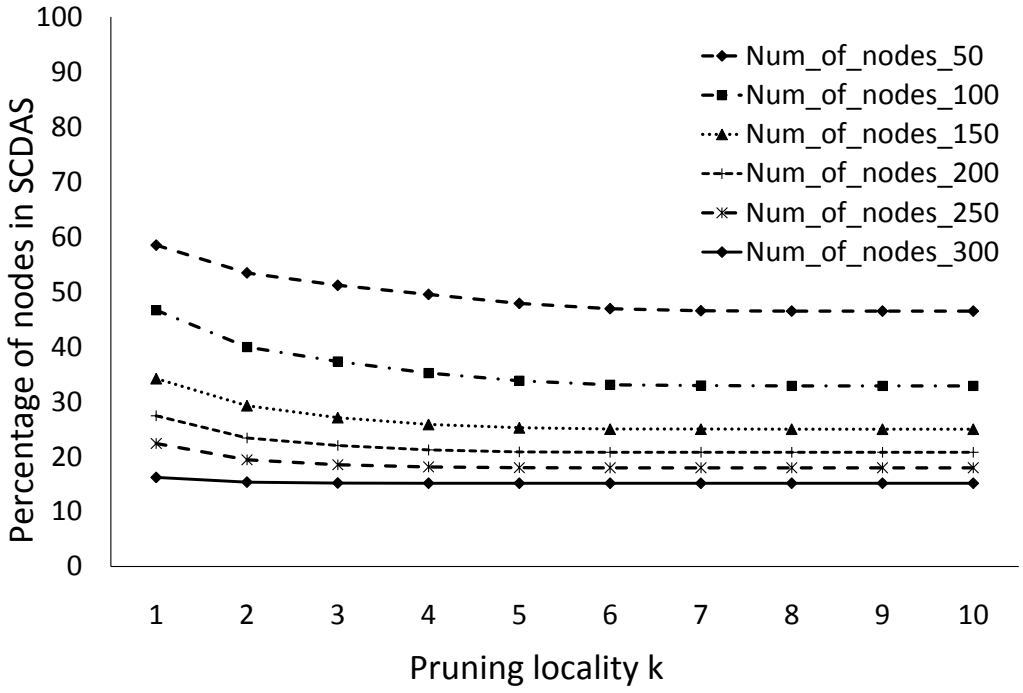


Figure 28: Impact of the locality of the strong $k-$connectivity test of the pruning procedure on the size of the SCDAS when transmission ranges are in $[10, 50]$

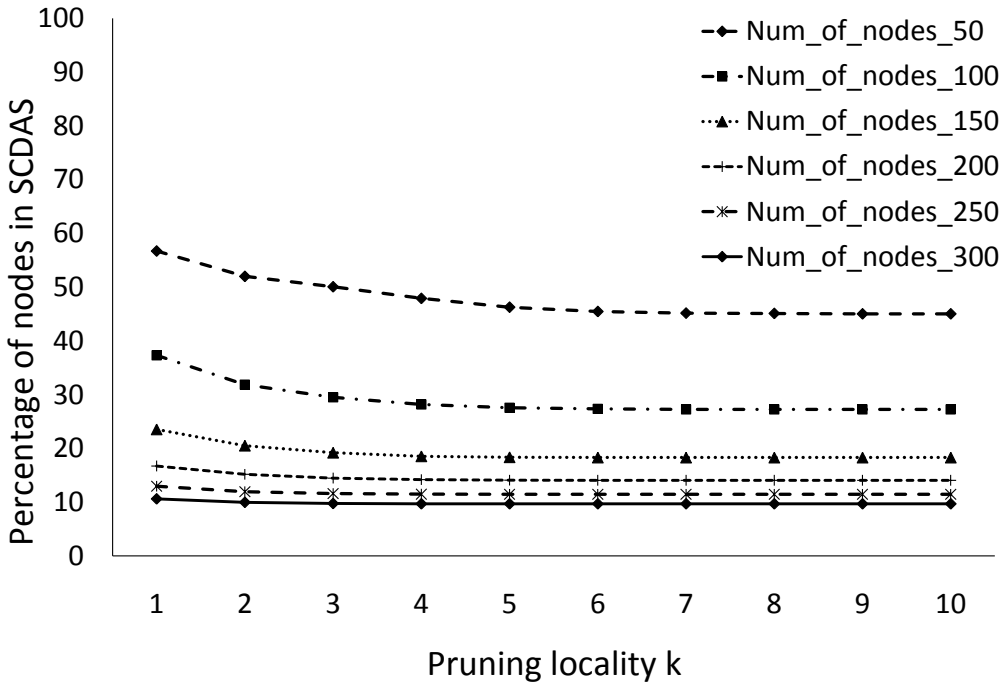


Figure 29: Impact of the locality of the strong $k-$connectivity test of the pruning procedure on the size of the SCDAS when transmission ranges are in $[20, 50]$

In denser networks, this number even decreases to 3. This behavior is expected since in denser networks, the expected number of nodes per hexagon increases and thus the initial SCDAS becomes sparse. It has been shown that as the percentage of unidirectional links decreases, increasing k would not result in significant decrease in the SCDAS size (See Figure 30,31,32).

In fact it is clear that in Figure 32 when all links are bidirectional the curve flattens after $k = 2$. Consequently, we have chosen $k = 4$ to be a good trade-off between the locality of the pruning phase of the algorithm and the gain that can be achieved by increasing k.

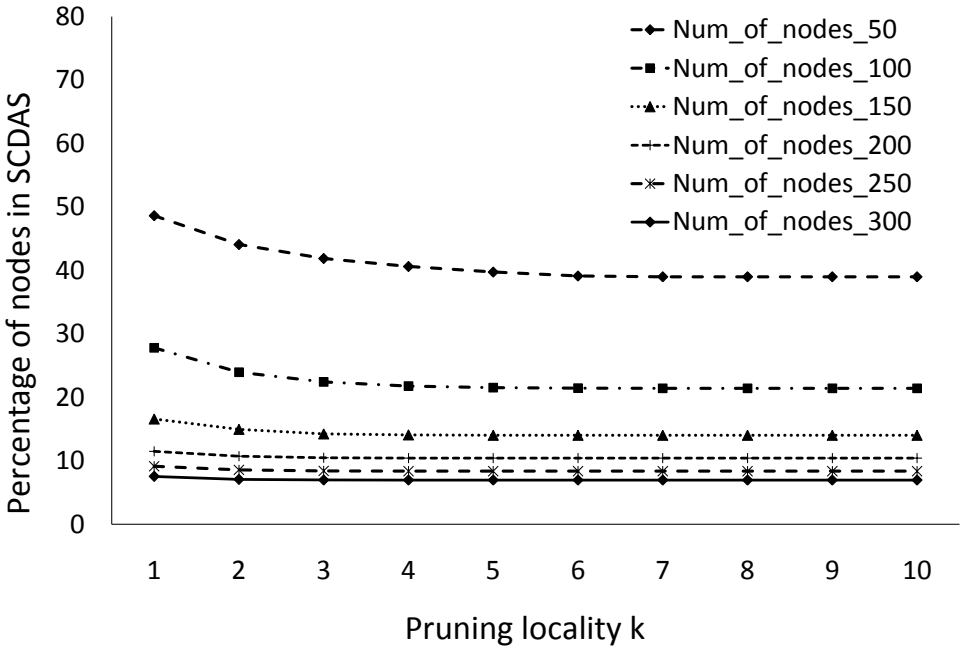


Figure 30: Impact of the locality of the strong $k-$connectivity test of the pruning procedure on the size of the SCDAS when transmission ranges are in $[30, 50]$

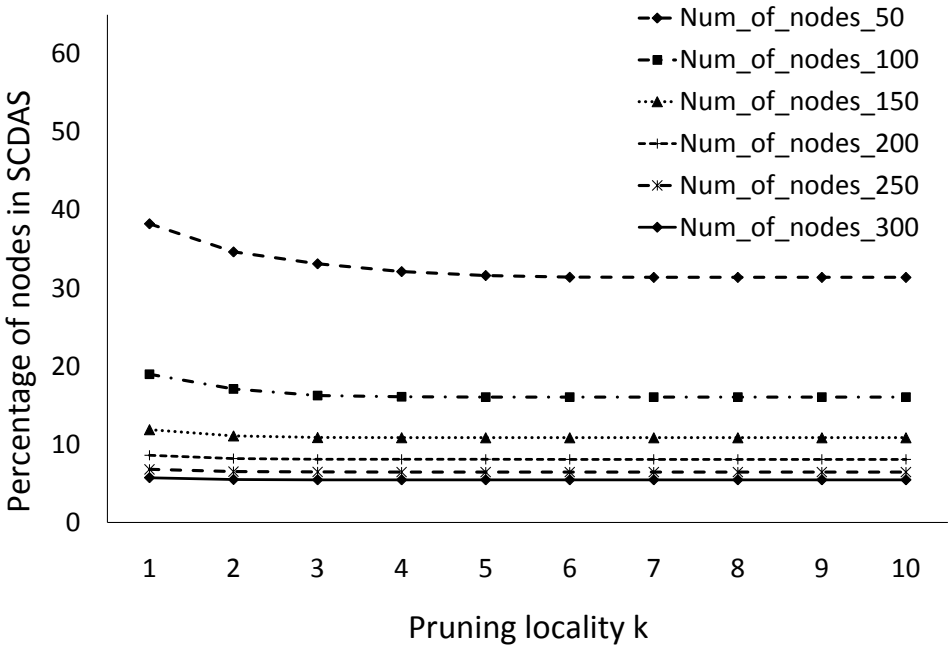


Figure 31: Impact of the locality of the strong $k-$connectivity test of the pruning procedure on the size of the SCDAS when transmission ranges are in $[40, 50]$

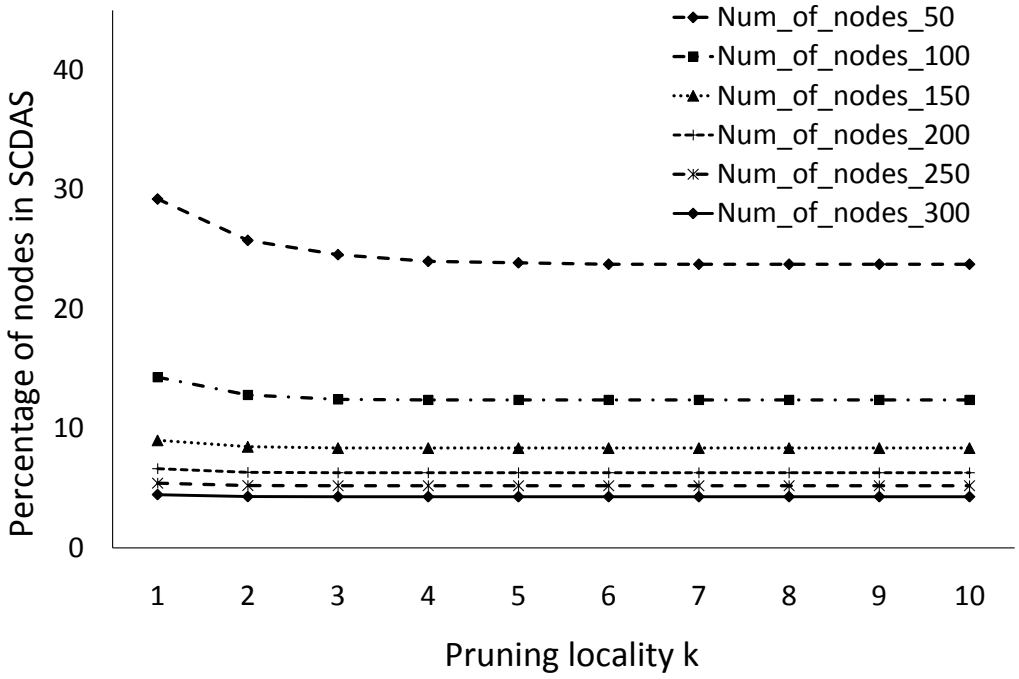


Figure 32: Impact of the locality of the strong $k-$connectivity test of the pruning procedure on the size of the SCDAS when transmission ranges are in $[50, 50]$

Therefore, in the performance evaluation of our algorithm, we selected two instances of our local algorithm with pruning localities 1 and 4 for the connectivity test of the pruning procedure, referred to as *TBL_1* and *TBL_4* respectively. We compare our algorithm with the few algorithms in the literature proposed for the construction of SCDAS in DGs, namely Wu's local algorithm [Wu02], hereafter referred to as *Wu* after the name the author, the two variant of the distributed algorithm in [KN10], referred by the authors as PInout_UD1 and PInout_UD4, and the two centralized algorithms in [PWW$^+$07], namely *Dominating-Absorbent Spanning Tree (DAST)* and *Greedy Strongly Connected Component Merging Algorithm (G_CMA)*. The extensive simulations in [KN10] show that PInout_UD1 and PInout_UD4 consistently outperforms the other algorithms. Thus, we only need to compare TBL_1 and TBL_4 with

PInout_UD1 and PInout_UD4. However, for completeness we have included the the results for Wu, DAST, and G_CMA in our figures. In our performance comparison, we focused on the impact of node density and the percentage of unidirectional links on the size of the SCDAS.

In order to show the impact of the percentage of unidirectional links independent of the node density, we have fixed the number of sensors and we investigated the effect of percentage of unidirectional links on graphs with 50, 100, 150, 200, 250 and 300 nodes separately. Our simulation results showed that the size of the SCDAS produced by the TBL algorithm is very close to the size of the SCDAS produced by the PInOut_UD in all cases. Thus, we only show the two extreme cases of very sparse networks and very dense networks when n is equal to 50 and 300 . These have been depicted in Figures 33 and 34 respectively. As it can be seen in Figure 33, our algorithm consistently outperforms all the other algorithms except for PInout_UD1 and PInout_UD4. Although TBL_4 and PInOut_UD4 generate SCDASs of almost the same size, TBL_1 generates a slightly smaller SCDAS than PInOut_UD1. It is noteworthy that despite the fact that TBL_1 and TBL_4 are local algorithms they perform as well as and sometimes even better than the distributed algorithms PInOut_UD1 and PInOut_UD4. As the number of nodes increases to 100 and 150 (moderate densities), the gap between DAST and Wu as one group and PInOut_UD1, PInOut_UD4, TBL_1 and TBL_4 as the other group widens. It can be seen in Figure 34 that as the percentage of unidirectional links decreases the gap between PInOut_UD1, PInOut_UD4, TBL_1 and TBL_4 becomes smaller and they all perform almost the same.

Analogously, to study the impact of node density on the size of the SCDAS, we fixed the percentage of unidirectional links and studied the effect the node densities for each set of graphs. As before, our simulations always showed that TBL and

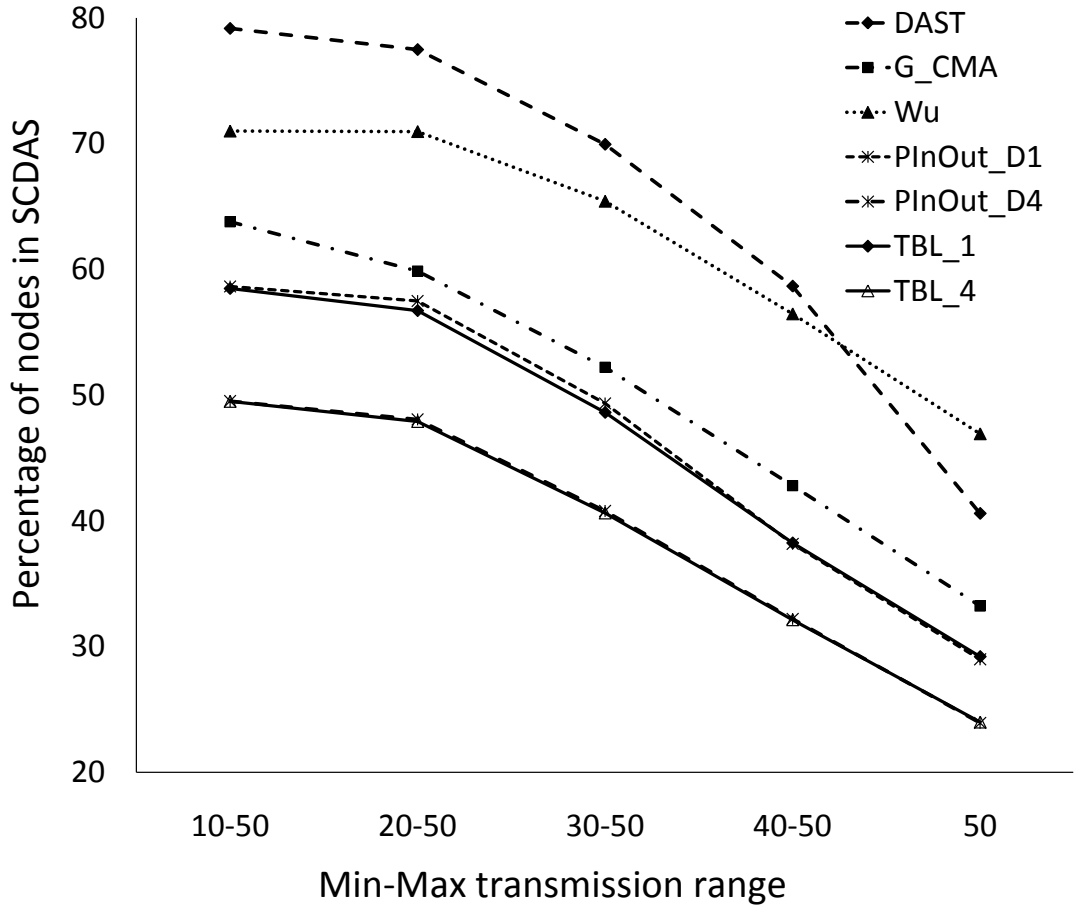


Figure 33: Impact of percentage of unidirectional links on the size of the SCDAS when number of nodes is 50

PInOut_UD produce SCDASs of almost the same size and for brevity we show the results for the two sets of graphs representing the sets with the maximum and minimum number of unidirectional links in Figures 35 and 36 respectively. Although TBL and PInOut_UD produce SCDASs of almost the same size, one can see that in very dense networks with high percentage of unidirectional links the difference between TBL_1 and PInOut_UD1 becomes even more significant (See Figure 35). In fact while the size of the SCDAS generated by PInOut_UD1 is 18.52% bigger than that of TBL_1, the size of the SCDAS produced by TBL_1 is only 1.5% bigger than PInOut_UD4. This implies that for dense networks with a high ratio of irregularities between the ratio of nodes' transmission ranges, TBL_1 performs as well as PInOut_UD4, which

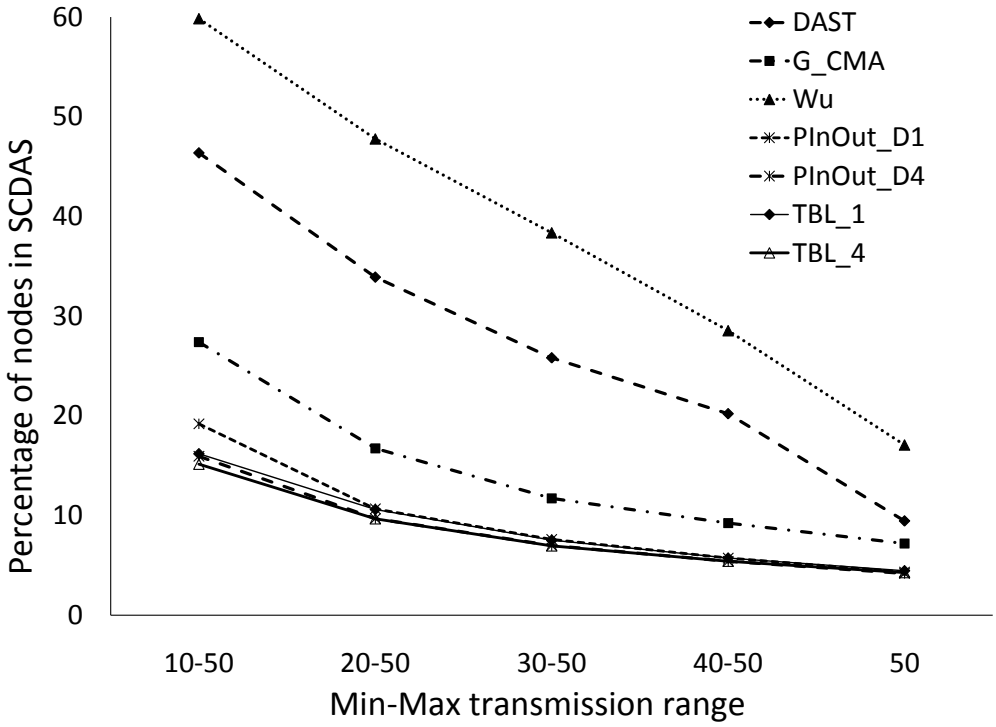


Figure 34: Impact of percentage of unidirectional links on the size of the SCDAS when number of nodes is 300

can significantly reduce the message complexity.

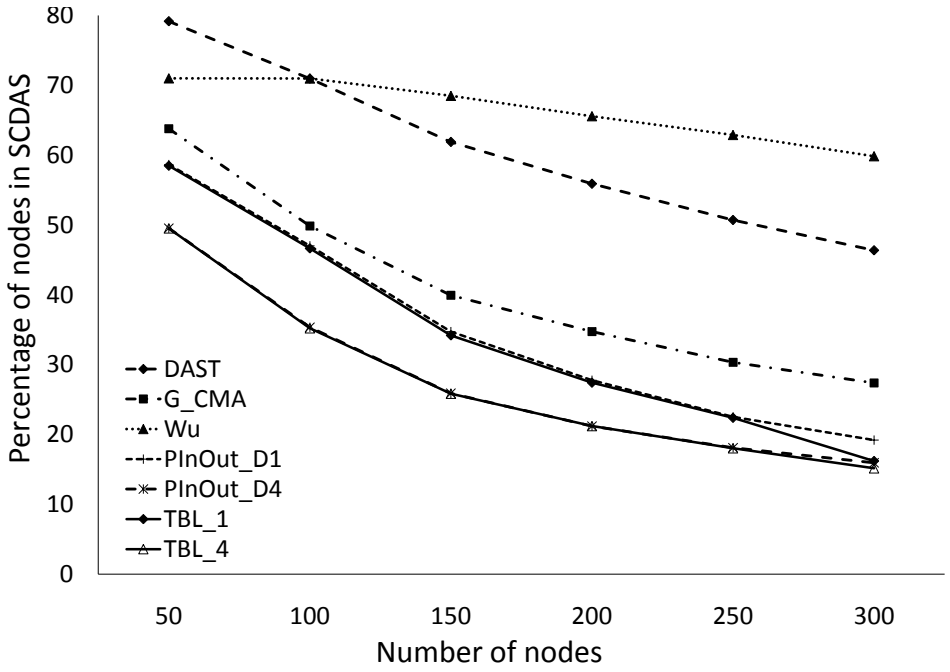


Figure 35: Impact of node density on the size of the SCDAS when transmission ranges are in $[10, 50]$

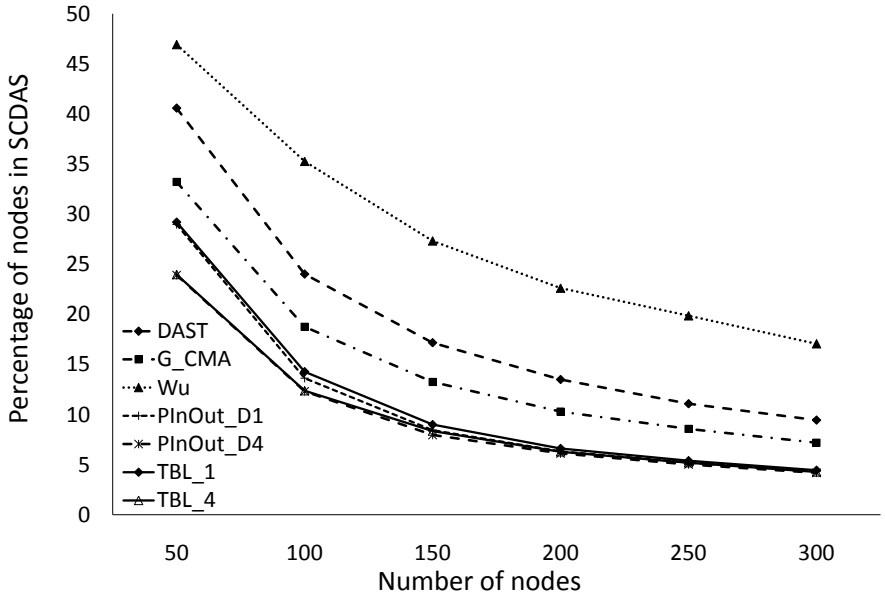


Figure 36: Impact of node density on the size of the SCDAS when transmission ranges are in $[50, 50]$

4.4 Conclusion

We proposed an efficient, local algorithm with constant approximation ratio for the construction of a strongly connected dominating and absorbent set in disk graphs. The SCDAS produced by our algorithm is significantly smaller compared to those produced by all the other SCDAS algorithms in the literature.

Chapter 5

Minimizing the Maximum Sensor Movement for Barrier Coverage

In this chapter, we consider generalizations of the *MinMax* problem studied in [CKK+09]. In Section 5.1, the previous studies for the intrusion detection problem with mobile sensors are described. In section 5.2, we discuss the problem of minimizing the maximum sensor movement *MinMax* for the coverage of line barriers studied in [CKK+09]. We then introduce two new generalizations for the MinMax problem: multiple barriers and circular barriers in Sections 5.3 and 5.4 respectively.

5.1 Related Work

Intrusion detection and border surveillance constitute a major application category for wireless sensor networks. A major goal in these applications is to detect intruders as they enter a region. This type of coverage is referred to as *barrier coverage*, where the sensors form a barrier for the intruders. Unlike the extensively studied problem of full coverage, [HT03, KLB04, MKPS01], where any point within the area is ensured to be covered by at least one sensor, in the barrier coverage problem only

the perimeter of the region is protected. This makes sense in applications involving boundary guard or movement detection where the intruder has to cross a boundary to enter the protected area, since full coverage requires many more sensors than the barrier coverage.

Furthermore, since some applications require deploying sensors in inhospitable terrains (e.g., forests, mountains, enemy regions), deployment of sensors in predetermined positions so as to achieve complete coverage is not always possible. Consequently, in these applications nodes are usually dispersed arbitrarily. In the literature, two approaches to attain complete barrier coverage are considered. In the first, sensor nodes are static and the dispersal is very dense around the boundary to ensure complete coverage. The studies using this approach either deal with the question of whether the barrier is completely covered or not, once the nodes are arbitrarily dispersed [CKL07, KLA05], or they estimate the density needed to achieve barrier coverage with a desired probability [BBSK07]. This approach needs many redundant sensors, leading to significant waste of sensors. In the second approach, sensors are mobile and once arbitrarily dispersed, they are instructed to move to final positions to achieve complete coverage. This is considered in [BBH$^+$08, CKK$^+$10, CKK$^+$09, SLX$^+$10, WCLP06], where some aspects of minimizing the energy consumption for the movements are studied. Since this chapter studies a generalization of the Min-Max problem studies in [CKK$^+$09], we discuss the results in [CKK$^+$09] thoroughly in Section 5.2. Below, the other studies are discussed with more details.

The authors of [WCLP06] considered the full coverage problem in a protected area and proposed protocols to calculate the target positions of the sensors so as to eliminate coverage holes (the area not covered by any sensor) . They used Voronoi diagrams to discover the coverage holes and designed three movement-assisted sensor deployment protocols, VEC (VECtorbased), VOR (VORonoi-based), and Minimax

based on the principle of moving sensors from densely deployed areas to sparsely deployed areas. For these three protocols, VEC pushes sensors away from a densely covered area, VOR pulls sensors to the sparsely covered area, and Minimax moves sensors to their local center area. Furthermore, they evaluated their protocols from aspects such as coverage, deployment time, moving distance, scalability to initial deployment and communication range through simulations.

In [BBH+08], the authors considered the barrier coverage problem on a circle and a simple polygon using n mobile sensors with identical sensing ranges. It is assumed that the the sum of sensors' coverages is equal to the length of the barrier to be covered. Also, each sensor has knowledge of the region to be barrier-covered, and of its geographic location. The authors first considered the scenario where sensors all lie on a line or on the perimeter of circle and they gave algorithms to assign final position to the sensors so as to achieve complete coverage while minimizing the sum of sensor movements, (*MinSum* problem). Then, they considered the case where sensors are lying in the interior of the polygon (circle), and they presented algorithms to assign final positions on the perimeter of the polygon (circle) so as to achieve complete coverage. They gave an $O(n^{3.5} \log n)$-time algorithm for the MinMax problem on a circle that moves the sensors to the perimeter of the circle to form a regular n-gon and an $O(mn^{3.5} \log n)$-time algorithm for the MinMax problem on a simple polygon, where m is the number of edges of the simple polygon. Furthermore, they studied the problem of minimizing the sum of movements of equal range sensors so as to achieve complete coverage of a circle and a simple polygon, and provided approximation algorithms for them. For a circle, they presented a PTAS and a $\pi + 1$ approximation algorithm with time complexities $O(\frac{1}{\epsilon}n^4)$ and $O(n^2)$, respectively. For a simple polygon, they presented a PTAS with running time of $O(\frac{1}{\epsilon}mn^5)$.

The intrusion detection problem in a thin strip $l * w$, where $l >> w$, is considered

in [SLX+10]. The authors assume that m mobile sensors and n stationary sensors have been arbitrarily dispersed in the thin strip area, and they studied the problem of moving the mobile sensors in order to form horizontal line barriers with the sensors along the thin strip area. They investigated the limit of the barrier coverage that a mobile sensor network can provide, as well as the requirement on the sensor mobility to reach the limit. Furthermore, they presented a sensor movement scheme to provide the maximum barrier coverage while minimizing the maximum moving distance among all sensors. In the presented scheme final positions assigned to sensors are coordinates on a grid, using a binary search with time complexity $O(\log lVE^2)$, where l is the area length and V and E are the number of vertices and edges on a graph G representing the UDG network formed by sensors.

In [CKK+10] the barrier coverage problem on a line segment barrier with mobile sensors have been studied. The mobile sensors are assigned final positions so as to achieve maximal coverage of the barrier while minimizing the sum of sensor movements. It is shown that the MinSum problem for sensors with non-identical ranges is NP-hard. For the case of sensors with identical ranges, the authors considered several scenarios depending on whether or not complete coverage is feasible. When the sum of sensors' coverages is less than the length of the area to be covered, complete coverage is not feasible, and they considered the maximal coverage problem. In case where complete coverage is not feasible, they distinguished two different problems depending on whether the coverage of sensors form a contiguous interval or not. For all the problems mentioned above, linear or quadratic algorithms were given.

5.2 The MinMax Problem on a Single Line Barrier

The problem of covering a line segment barrier with wireless mobile sensors was considered in [CKK+09], where sensors are initially placed arbitrarily on the same

line. The goal is to move the sensors to their final position so that maximum coverage is established while the maximum displacement of any sensor is minimized (*MinMax*). Three variants of the problem, based on (i) whether or not complete coverage is possible and (ii) in the case when complete coverage is impossible, whether or not the maximal coverage is required to be contiguous, have been studied.

If the total coverage of all sensors R is either greater than or equal to the length of the barrier L, then complete coverage is feasible. The *MinMax optimization problem* for $R \geq L$ is defined as follows.

MinMax optimization problem:

$$\text{minimize } \{\max_{1 \leq i \leq n} |m_i|\} \text{ subject to } [0, L] \subseteq \bigcup_{i=1}^{n} [x_i - r_i, x_i + r_i]$$

Where, m_i is the distance S_i has traveled from its initial position. A movement to the left will be denoted by $m_i \leq 0$ and movement to the right by $m_i \geq 0$.

When $R < L$ and thus complete coverage of $[0, L]$ is not feasible, they introduced the problem of finding an arrangement of sensors that attains the largest possible coverage while at the same time minimizing the maximum movement of sensors. They consider two variants of the optimization problem referred to as the *non-contiguous MinMax optimization problem* and the *contiguous MinMax optimization problem* for $R < L$, defined below.

Non-contiguous MinMax optimization problem:

$$\text{minimize } \{\max_{1 \leq i \leq n} |m_i|\} \text{ subject to } \bigcup_{i=1}^{n} [x_i - r_i, x_i + r_i] \subseteq [0, L] \text{ and}$$
$$|\bigcup_{i=1}^{n} [x_i - r_i, x_i + r_i]| = R.$$

Contiguous MinMax optimization problem:

$$\text{minimize } \{\max_{1\leq i\leq n} |m_i|\} \text{ subject to } \bigcup_{i=1}^{n} [x_i - r_i, x_i + r_i] \subseteq [0, L] \text{ and}$$

$$|\bigcup_{i=1}^{n} [x_i - r_i, x_i + r_i]| = R \text{ and}$$

$$\bigcup_{i=1}^{n} [x_i - r_i, x_i + r_i] \text{ is an interval.}$$

All the algorithms given for sensors with equal ranges use the following lemma, referred to as the *order-preservation lemma*.

Lemma 5.2.1. ***(Order Preservation)***. *Let $S_1, S_2, \ldots, S_n$ be sensors with ranges $r_1, r_2, \ldots, r_n$ in initial positions $x_1 \leq x_2 \leq \ldots x_n$. If there are no two sensors S_i and S_j , $1 \leq i \neq j \leq n$ such that $x_j - r_j < x_i + r_i$ and $x_j + r_j > x_i + r_i$ then there is an order-preserving optimal solution of any of the three versions of the MinMax optimization problem.*

Note that the condition of the above lemma is clearly satisfied when the covering intervals of the sensors form a proper interval graph, where the interval graph is a graph in which sensor ranges represent the vertices and a proper interval graph is an interval graph in which no interval properly contains another (see [Fis85]). Consequently, the condition of the lemma always satisfies in the case of sensors with identical ranges.

When the sensors have unequal ranges, if the covering intervals of the sensors form a proper interval graph, all the algorithms in Table 1 are still valid. However, it is an open problem whether or not the MinMax optimization problem is NP-complete in general. The MinMax problem whereby one of the sensors is assigned a fixed position is shown to be NP-complete.

Table 1 shows the results for the MinMax problem when sensors have identical

ranges, L is the length of the barrier and R the sum of length of covering intervals.

Table 1: Summary of time complexities of the algorithms in [CKK$^+$09], where sensors have identical ranges

		contiguous	non-contiguous
$R < L$		$O(n)$	$O(n)$
$R = L$		$O(n)$	N.A.
$R > L$	optimal	$O(n^2)$	N.A.
	$2-$approximation	$O(n)$	N.A.
	$1+\epsilon-$approximation	$O(n \log \frac{\log(C/g)}{\log(1+\epsilon)})$	N.A.

5.3 The MinMax Problem with Multiple Barriers

In this section, we consider the problem of covering multiple barriers with a set of n sensor nodes with identical sensing range r. The barriers are disjoint line segments with lengths $L_1, L_2, \ldots, L_m$. We consider all possible scenarios: $R = \sum_{i=1}^{m} L_i$, $R < \sum_{i=1}^{m} L_i$ and $R > \sum_{i=1}^{m} L_i$, and we peresent centralized polynomial algorithms for all of them. For each of the mentioned scenarios, we first study the corresponding problem on two barriers, and then we generalize the solution to the case of m barriers.

The shift value of sensor S_i is its displacement value, d_i, from its initial position. The left shifts are shown by negative values, and right shifts by positive values. The maximum shift is found by comparing the absolute values of shifts for all sensors, i.e. $Max_{i\in\{1,..,n\}}|d_i|$.

We generalize the ordering lemma for the case of multiple barriers.

Lemma 5.3.1. (Order preservation on multiple barriers) *Let $S_1, S_2, ..., S_n$ be sensors with identical sensing range r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Also, let $B_1, B_2, ..., B_m$ be disjoint line segments on the same infinite line, with lengths*

$L_1, L_2, ..., L_m$. *There is an order−preserving optimal solution of all three versions of the MinMax optimization problem on m barriers.*

Proof. Consider a solution of a MinMax problem $y_1, y_2, ..., y_n$, where y_i is the final position of S_i, in which a sensor S_i preceding S_j in initial order, succeeds S_j in final position; i.e., there are sensors S_i and S_j, where $x_i < x_j$ and $y_i > y_j$. The shift of sensors S_i and S_j are $|y_i - x_i|$ and $|y_j - x_j|$ respectively. Swapping S_i and S_j in the final position would result in shifts $|y_j - x_i|$ and $|y_i - x_j|$. It can be easily seen that we can swap these two sensors without increasing the value of the maximal move while covering the same area in the solution. Therefore by a sequence of switches we can obtain an optimal solution that preserves the original order of sensors. ■

Furthermore, unlike the MinMax optimization problem for different sensor ranges on a single barrier where sensor ranges form a proper interval graph, the order preservation lemma does not generalize for multiple barriers with arbitrary range sensors. In fact, the order preservation lemma does not hold as soon as we increase the number of barriers to two. Consider the following example. There are two sensors S_1 and S_2, at initial positions 2 and 8 with transmission ranges 2 and 1 respectively. Also, we have two barriers $B_1 = [0, 2]$ and $B_2 = [6, 10]$. The only solution that provides complete coverage of B_1 and B_2 is to move S_2 to final position 1, and S_1 to final position 8. Clearly this does not preserve the original order of the sensors.

Furthermore, we show that when sensors have arbitrary ranges, the MinMax optimization problem on m barriers is NP-hard. In fact, below we show that even with two barriers the problem remains NP-hard.

Theorem 5.3.2. *Let $S_1, S_2, ..., S_n$ be sensors with sensing ranges $r_1, r_2, ..., r_n$ in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Also, let B_1 and B_2 be disjoint line segments on the same infinite line, with lengths L_1 and L_2 respectively. The MinMax optimization problem on two barriers is NP-hard.*

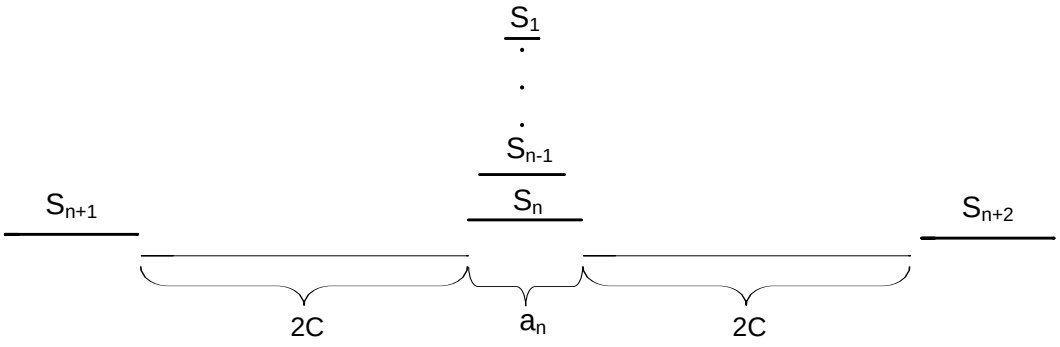


Figure 37: Arrangements of sensors for proving the NP-completeness of the MinMax optimization problem on two barriers when sensors have arbitrary ranges.

Proof. We reduce the *partition problem* [GJ90] into the MinMax optimization problem on two barriers. The partition problem is defined as follows:

Given a sequence of integers $a_1 \leq a_2 \leq ... \leq a_n$, determine whether there exists a set of indices J such that $\sum_{i \in J} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$.

Let $C = \frac{1}{2} \sum_{i=1}^{n} a_i$. Given an instance of the partition problem, we transform it to the MinMax optimization problem on two line segments for the sensor set $S = \{S_1, S_2, ..., S_n, S_{n+1}, S_{n+2}\}$. The sensors $S_1, S_2, ..., S_n$ have sensing ranges $\frac{a_1}{2} \leq \frac{a_2}{2} \leq ... \leq \frac{a_n}{2}$ and have initial positions $2C + \frac{a_n}{2}$. Furthermore, sensors S_{n+1}, S_{n+2} with sensing range $\frac{C}{2}$ are at initial positions $\frac{-C}{2}$ and $a_n + \frac{9C}{2}$ (see Figure 37).

Now if there is a set of indices J such that $\sum_{i \in J} a_i = C$, there is a solution to the barrier coverage problem such that for any $i \in J$ the sensor S_i is moved to B_1 and for any $i \notin J$ the sensor S_i is moved to B_2. Thus, this way we can cover regions of size C on both B_1 and B_2 with all shifts being at most of size C. The half left of B_1

and half right of B_2 can be covered by S_{n+1} and S_{n+2} with shifts being at most C

If such a partition does not exist, then any distribution of sensors in S covers a region of size less that C either on B_1 or B_2. Therefore, we have to move one of the sensors S_{n+1} or S_{n+2} more than C to get a solution.

Thus, if there is an algorithm that can determine if there are movements of sensors on two line segments so as to achieve maximum coverage while the maximum movement of any sensors is at most C, we can determine whether the partition problem has a solution. Clearly, the transformation from the partition problem to the sensor movement problem is polynomial. ■

In view of theorem 5.3.2, through the rest of this section, we always assume that nodes have identical sensing ranges.

5.3.1 $L_1 + L_2 = 2rn$

Theorem 5.3.3. *Let $S_1, S_2, ..., S_n$ be sensors with identical sensing range r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Also, let B_1 and B_2 be two disjoint line segments with lengths L_1 and L_2 respectively; i.e. $B_1 = [0, L_1]$ and $B_2 = [L_1 + g, L_1 + L_2 + g]$, where g is the gap length between the two barriers B_1 and B_2, and $L_1 + L_2 = 2rn$. There is an $O(n^2)$ algorithm that solves the MinMax optimization problem of covering two line segments B_1 and B_2, so that the maximal value of the shift of any sensor is minimized.*

Proof. We need $\lceil \frac{L_1}{2r} \rceil + \lceil \frac{L_2}{2r} \rceil$ sensors to cover L_1 and L_2 completely. Since $L_1 + L_2$ is a multiple of $2r$, either both L_1 and L_2 are multiples of $2r$ or neither L_1 nor L_2 is a multiple of $2r$. In the former, $\lceil \frac{L_1}{2r} \rceil + \lceil \frac{L_2}{2r} \rceil$ is equal to n, and complete coverage is possible. The later implies that $\lceil \frac{L_1}{2r} \rceil + \lceil \frac{L_2}{2r} \rceil$ is $n + 1$, indicating that complete coverage is not feasible. Thus, we consider two scenarios:

1. Both L_1 and L_2 are multiples of $2rn$; i.e. $\exists i \in \mathbb{N}, \exists j \in \mathbb{N}, L_1 = 2ri \wedge L_2 = 2rj$. In this case complete coverage is possible, and all sensors have predetermined positions. Move senors S_1, S_2, ... ,S_i to positions $le(B_1) + r$, $le(B_1) + 3r$, ..., $le(B_1) + (2i - 1)r$ and sensors S_{i+1}, S_{i+2}, ... ,S_n to positions $le(B_2) + r$, $le(B_2) + 3r$, ..., $le(B_2) + (2j - 1)r$. Clearly, this can be done in $O(n)$ time.

2. Neither L_1 nor L_2 are multiples of $2rn$; i.e. $\nexists i \in \mathbb{N}, \nexists j \in \mathbb{N}, L_1 = 2ri \wedge L_2 = 2rj$. Let n_1 and n_2 be the minimum number of sensors needed to completely cover B_1 and B_2 respectively; i.e. $n_1 = \lceil \frac{L_1}{2r} \rceil$ and $n_2 = \lceil \frac{L_2}{2r} \rceil$. Furthermore, let f_1 and f_2 be $L_1 - 2r(n_1 - 1)$ and $L_2 - 2r(n_2 - 1)$ respectively. Since, n_1+n_2 is equal to $n + 1$, and we only have n sensors, we aim at achieving maximum possible coverage while minimizing the maximum sensor movement. Using Lemma 5.3.1, sensors $\{S_1, S_2, \ldots, S_{n_1-1}\}$ are used toward coverage of B_1 and sensors $\{S_{n_1+1}, S_{n_1+2}, \ldots, S_n\}$ are used toward coverage of B_2 . In order to achieve maximum coverage S_{n_1} can cover B_1 , B_2 or a part of B_1 and a part of B_2 at the same time providing coverage $2rn - f_2$, $2rn - f_1$ and $2rn - g$ respectively. Note that the coverage $2rn - g$ is only feasible when $g < 2r$. We distinguish three possible scenarios.

   (a) $f_1 = min(g, f_1, f_2)$: maximum coverage is achieved by using $\{S_{n_1}, S_{n_1+1}, \ldots, S_n\}$ to cover B_2 completely, and using sensors in $\{S_1, S_1, ..., S_{n_1-1}\}$ to cover B_1 partially. The former can be achieved in $O(n^2)$ using the algorithm in [CKK$^+$09] for $R > L$, and the later can be achieved in $O(n)$ using the algorithm in [CKK$^+$09] for $R < L$.

   (b) $f_2 = min(g, f_1, f_2)$: maximum coverage is achieved by using $\{S_1, S_1, ..., S_{n_1}\}$ to cover B_1 completely, and using sensors in $\{S_{n_1+1}, S_{n_1+2}, \ldots, S_n\}$ to cover B_2 partially. The former can be achieved in $O(n^2)$ using the algorithm in

[CKK+09] for $R > L$, and the later can be achieved in $O(n)$ using the algorithm in [CKK+09] for $R < L$.

(c) $g = min(g, f_1, f_2)$: maximum coverage is achieved only if S_{n_1} completely covers the gap (covers a part of B_1 and a part of B_2 at the same time). We show that any optimal solution that minimizes the maximum sensor movement while maximizing the coverage on virtual barrier $[0, L_1 + L_2 + g]$ has a sensor that completely covers $[L_1, L_1 + g]$. Assume there is an optimal solution with maximal coverage $2rn$ on $[0, L_1 + L_2 + g]$ with no sensor that completely covers the gap. Therefore, the number of sensors that are to the left of $L_1 + g$ is less than or equal to $\lfloor \frac{L_1+g}{2r} \rfloor = \lfloor \frac{2rn_1-2r+f_1+g}{2r} \rfloor = n_1 + \lfloor \frac{f_1+g-2r}{2r} \rfloor$, and the number of sensors that are to the right of L_1 is less than or equal to $\lfloor \frac{L_2+g}{2r} \rfloor = \lfloor \frac{2rn_2-2r+f_2+g}{2r} \rfloor = n_2 + \lfloor \frac{f_2+g-2r}{2r} \rfloor$.

Note that since $g = min(g, f_1, f_2)$, either $g < r$ or $f_1 = f_2 = r$. In the later, using S_{n_1} toward coverage of B_1, B_2 or a part of B_1 and a part of B_2, provides the same coverage of $2rn - r$, and therefore return the solution that minimizes the maximum movement. In the later, both f_1+g and f_2+g are less than $2r$ and therefore $n_1 + \lfloor \frac{f_1+g-2r}{2r} \rfloor = n_1 - 1$ and $n_2 + \lfloor \frac{f_2+g-2r}{2r} \rfloor = n_2 - 1$. This means that there are maximum $n_1 - 1 + n_2 - 1 = n - 1$ sensors used to cover the barrier $[0, L_1 + L_2 + g]$ which contradicts the fact that the maximal coverage is equal to $2rn$. Therefore, any optimal solution that minimizes the maximum sensor movement while maximizing the coverage on virtual barrier $[0, L_1 + L_2 + g]$ has a sensor that completely covers $[L_1, L_1 + g]$, and this can be obtained in $O(n)$ using the algorithm in [CKK+09] for $R < L$.

■

Corollary 5.3.4. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Let $B_1, B_2, \ldots B_m$ be m line segments on the same infinite line with lengths $L_1, L_2, \ldots, L_m$. Let $n_i > 0$ be the minimum number of sensors that can be fully contained in B_i, $n_i = \lfloor \frac{L_i}{2r} \rfloor$, and let $f_i = L_i - 2rn_i$. Furthermore, let g_i be the length of the gap between L_i and L_{i+1}. There is an $O(n^2 + mn)$ algorithm that solves the MinMax optimization problem of covering m line segments B_1 and B_2, ..., B_m with $\sum_{i=1}^{m} |B_i| = 2rn$ so that the maximal value of the shift of any sensor is minimized.*

Proof. If all the barriers are multiple of $2r$, then the final positions are predetermined and the optimal solution can be obtained in linear time. Otherwise, let f be $\max_{1 \leq i \leq m}(f_i)$. Let k be the number of barriers with $f_i = f$. Clearly, k is less than m. For every barrier B_i with $f_i = f$, overcover B_i in $O(n_i^2)$ and the undercover the rest of barriers in $O(n - n_i)$. This can be done in $O(n^2 + mn)$. Furthermore, when covering a gap g_i provides more coverage, covering the combined barrier takes $O(n_i + n_{i+1})$ time and since there are at most $\frac{m}{2}$ combined barriers that adds at most another $O(mn)$ resulting in total time complexity $O(n^2 + mn)$. ■

5.3.2 $L_1 + L_2 > 2rn$

It is obvious that when the combined length of the barriers is greater than the total sensor coverage $2rn$, complete coverage is not possible. Therefore, we consider the problem of maximizing the possible coverage, while minimizing the maximum sensor movement.

Let n_1 and n_2 be the maximum number of sensors that can be fully contained in B_1 and B_2 respectively; i.e. $n_1 = \lfloor \frac{L_1}{2r} \rfloor$ and $n_2 = \lfloor \frac{L_2}{2r} \rfloor$. Clearly, if $n_1 + n_2 \geq n$, maximal coverage of $2rn$ is feasible. Using Lemma 5.3.1, the problem reduces to the problem of partitioning the set of sensors into two sets $N_1 = \{S_1, S_2, ..., S_j\}$ and

$N_2 = \{S_{j+1}, S_{j+2}, ..., S_n\}$, and using the sensors in N_1 and N_2 toward coverage of B_1 and B_2 respectively. We can the use the linear algorithm is provided in [CKK$^+$09] for the case where $R < L$ to cover both B_1 and B_2. Throughout this study, we use the index of the last sensor in N_1 to represent the two partitioned sets and refer to it as the *cutting point* j.

It should be noted that when $L_1 + L_2$ is just slightly bigger than $2rn$, i.e. $n_1 + n_2 = n - 1$, the coverage of $2rn$ is not possible, and we should consider this case separately.

Lemma 5.3.5. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. After solving the two MinMax problems for B_1 with sensor set $N_1 = \{S_1, S_2, ..., S_j\}$ and B_2 with sensor set $N_2 = \{S_{j+1}, S_{j+2}, ..., S_n\}$, let $rs(B_1)$, $ls(B_1)$, $rs(B_2)$, and $ls(B_2)$ be the maximum right shift in N_1, the maximum left shift in N_1, the maximum right shift in N_2, and the maximum left shift in N_2 respectively. Then,*

i) If $rs(B_1) = Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\}$, the solution is optimal.

ii) If $|ls(B_1)| = Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\}$, none of the cutting points k, where $k > j$, is better than j.

iii) If $rs(B_2) = Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\}$, none of the cutting points k, where $k < j$, is better than j.

iv) If $|ls(B_2)| = Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\}$, the solution is optimal.

Proof. i) In order to show that if $rs(B_1) = Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\}$, the current solution is optimal, we show that there is no other cutting point with a smaller right shift than the current cut. First we show that choosing a cutting point k, with $k < j$ would not give a better solution. Then, we show that the right shift cannot be reduced by choosing a cutting point k, with $k > j$.

(a) A cutting point k, with $k < j$ either shifts the sensors in $\{S_{k+1}, S_{k+2}..., S_j\}$ further to the right or does not move them at all and thus adding a value $c_i \geq 0$ to all the D_is. Since c_is are non$-$negative, the maximum shift cannot be reduced.

(b) The optimal solution for the MinMax problem with the sensor set $\{S_1, S_2, ..., S_j\}$ on B_1 is obtained by first solving the problem on an infinite line. The algorithm for the infinite line would only shift the sensors so that the sensing range of no two sensors intersect. Since $L_1 + L_2 > 2rn$ all the sensors have to used for the coverage and thus these shift values cannot be reduced. Once the problem is solved for the infinite line, we check if all sensors are contained in B_1. If they are already in B_1, then the MinMax problem with the sensor set $\{S_1, S_2, ..., S_j\}$ is solved and none of the shifts including $rs(B_1)$ can be reduced further. If sensors $\{S_1, S_2, ..., S_m\}$ are to the left of the left point of $B1$, $le(B_1)$, then S_1,S_2,...,S_m have to move to $le(B_1)+r$, $le(B_1)+3r$,..., $le(B_1) + (2m_1)r$ respectively and thus incrementing the right shifts values. Again, since all the sensors have to be used this cannot be avoided either.

So the value of right shift cannot be reduced by choosing a different cutting point and therefore the solution obtained by choosing S_j as a cutting point is an optimal solution.

ii) If $|ls(B_1)| = Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\}$, any cutting point k, where $k > j$, would shift sensors $\{S_1, S_2, ..., S_j\}$ further to the left and thus adding the shift value $c_i \leq 0$ to all D_is for $i \in \{1, 2, ..., j\}$. Since $ls(B_1)$ is non$-$positive, $|ls(B_1)|$ cannot further be reduced, thus none of the cutting points k, where $k > j$ is a better solution than S_j.

iii) If $rs(B_2) = Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\}$, any cutting point k, where $k < j$, would shift sensors $\{S_{k+1}, S_{k+2}, ..., S_n\}$ further to the left and thus adding the shift value $c_i \leq 0$ to all D_is for $i \in \{j+1, j+2, ..., n\}$. Since $rs(B_2)$ is non−negative, $rs(B_2)$ cannot further be reduced, and so, none of the cutting points k, where $k < j$ is a better solution than S_j.

iv) Similar to the proof for part (i), we first show that choosing a cutting point k, with $k > j$ would not give a better solution. Then, we show that the left shift cannot be reduced by choosing a cutting point k, with $k < j$.

   (a) Choosing a cutting point k, with $k > j$ would shift sensors $\{S_1, S_2, ..., S_k\}$ to the left and thus adding non−positive values to all D_is for $i \in \{j+1, j+2, ...k\}$, and since $ls(B_2)$ is a non−positive value as well, the left shift $|ls(B_2)|$ cannot be reduced by choosing a cutting point k, with $k > j$.

   (b) Similar to (i)-b, if after finding the optimal solution for the set of sensors N_2 on the infinite line, there are still sensors S_m, S_{m+1}, ..., S_n to the right of B_2, $re(B_2)$, they have to go to positions $re(B_2) - [2(n-m)+1]r$, $re(B_2) - [2(n-m-1)+1]r$, ..., $re(B_2) - r$, respectively. Therefore, the value of the left shift cannot be reduced by choosing a different cutting point.

Therefore, the solution obtained by choosing S_j as a cutting point is an optimal solution.

■

Theorem 5.3.6. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$ with $L_1 + L_2 > 2rn$. There is an $O(n \log n + n^2)$ algorithm that solves the MinMax optimization problem of covering two line segments B_1 and B_2 so that the maximal value of the shift of any sensor is minimized.*

Proof. Let n_1 and n_2 be the maximum number of non−intersecting sensors that can be contained on barriers B_1 and B_2, respectively; i.e. $n_1 = \lfloor \frac{L_1}{2r} \rfloor$ and $n_2 = \lfloor \frac{L_2}{2r} \rfloor$. Furthermore, if $n_1 > n$ ($n_2 > n$), let $n_1 = n$ ($n_2 = n$). The algorithm would then find an optimal cutting point j, using a binary search, for which the maximum movement for the MinMax problems, $\{S_1, S_2, ..., S_j\}$ on B_1 and $\{S_{j+1}, S_{j+2}, ..., S_n\}$ on B_2, is minimized.

Unlike the case where $n_1 + n_2 < n$, when $n_1 + n_2 \geq n$, $2rn$ coverage is possible . We consider three different cases depending on the values of n_1 and n_2.

i) $n_1 + n_2 = n - 1$. This case is similar to the case where $L_1 + L_2 = 2rn$, and can be solved in $O(n^2)$ using the algorithm proposed in Section 6.2.1.

ii) $n_1 + n_2 = n$. Then the only possible solution is to solve the MinMax problem on B_1 for $\{S_1, S_2, ..., S_{n_1}\}$ and on B_2 for $\{S_{n_1+1}, s_{n_1+2}, ..., S_n\}$ using the algorithm in [CKK+09] for $R > L$ with time complexity $O(n^2) + O(n^2) = O(n^2)$.

iii) $n_1+n_2 > n$. In this case $2rn$ coverage is possible and we can consider the problems of covering barrier B_1, and B_2 independently. Since the maximum number of sensors that can be contained on B_1 without intersecting is n_1, the possible cuts are in $N_1 = \{1, 2, ..., n_1\}$. Similarly, in order to avoid having more than n_2 sensors on B_2, the possible cuts should belong to $N_2 = \{n - (n_2 - 1), n - (n_2 - 2)..., n\}$ as well. Since $n_1 + n_2 > n$, then $N_1 \cap N_2 \neq \emptyset$. If neither n_1 nor n_2 are equal to n, let l be $n_1 + n_2 - n$, otherwise let l be $\min\{n_1, n_2\}$. The candidate set for the valid cutting points $C = N_1 \cap N_2$ would then $\{n_1 - (l - 1), ..., n_1 - 1, n_1\}$. The algorithm would then take i the middle element in C, $i = (n_1 - (l-1) + n_1)/2$, as a cutting point and solves the two MinMax problems in $O(n)$ for the sensors sets $\{S_1, S_2, ..., S_i\}$ and $\{S_{i+1}, S_{i+2}, ..., S_n\}$ on B_1 and B_2, respectively. Depending on values of $rs(B_1)$, $ls(B_1)$, $rs(B_2)$, $ls(B_2)$, and using Lemma 5.3.5, either the cut is

the optimal solution or the optimal cut is in $C_1 = \{n_1 - (l-1), n_1 - (l-2), ..., i\}$ or $C_2 = \{i+1, i+2, ..., n_1\}$. Let C be the subset containing the optimal cut and repeat the same procedure. Since we are reducing the set size, which contains the optimal cut, by half each time, the algorithm has time complexity $O(\log n)$ and knowing that $l \leq n$, the algorithm has time complexity $O(n \log n)$.

■

Corollary 5.3.7. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(\max(n(\log n)^k, n^2 + mn))$ algorithm that solves the MinMax optimization problem of covering $m = 2^k$ line segments B_1 and B_2, ..., B_m with $\sum_{i=1}^{m} |B_i| > 2rn$ so that the maximal value of the shift of any sensor is minimized.*

Proof. If coverage of $2rn$ is not possible, the problem is similar to the case where $\sum_{i=1}^{m} |B_i| = 2rn$ and can be solved in $O(n^2 + mn)$ time. Otherwise the cutting points can be found using binary search in $O((\log n)^k)$, and the problem can be solved in $O((\log n)^k)$. Thus, the solution can be found with total time complexity $O(\max(n(\log n)^k, n^2 + mn))$. ■

5.3.3 $L_1 + L_2 < 2rn$

In this scenario, since the sum of the the barrier lengths is smaller than the maximal coverage provided by all sensors, not all sensors need to participate in the coverage problem. Therefore, we need to move some of the sensors in order to cover the two barriers while minimizing the maximum movement. In the case of single barrier, an $O(n^2)$ algorithm for an optimal solution is provided in [CKK+09]. We show that the following lemma holds, in order to use the algorithm presented in [CKK+09] for the case of two barriers.

Lemma 5.3.8. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Let j be any number in $\{1, 2, ..., n\}$, such that $2rj \geq L_1$ and $2r(n-j) \geq L_2$. After solving the the two MinMax problems for B_1 with sensor set $N_1 = \{S_1, S_2, ..., S_j\}$ and B_2 with sensor set $N_2 = \{S_{j+1}, S_{j+2}, ..., S_n\}$, let $rs(B_1)$, $ls(B_1)$, $rs(B_2)$, and $ls(B_2)$ be the maximum right shift in N_1, the maximum left shift in N_1, the maximum right shift in N_2, and the maximum left shift in N_2 respectively. The followings are true:*

i) If $Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\} = \{|rs(B_1)| \vee |rs(B_2)|\}$, none of the cutting points k, where $k < j$, is a better solution than S_j.

ii) If $Max\{|rs(B_1)|, |ls(B_1)|, |rs(B_2)|, |ls(B_2)|\} = \{|ls(B_1)| \vee |ls(B_2)|\}$, none of the cutting points k, where $k > j$, is a better solution than S_j.

Proof. i) If the maximum shift occurred is a right shift on B_1 (B_2), any cutting point k, with $k < j$ shifts the sensors in $\{S_{k+1}, S_{k+2}..., S_j\}$ ($\{S_j, S_{j+1}, ..., S_n\}$) further to the right and thus adding a value $c_i \geq 0$ to all the D_is. Since c_is are non$-$negative, the maximum shift cannot be reduced.

ii) If the maximum shift occurred is a left shift on B_1 (B_2), any cutting point k, where $k > j$, would shift sensors $\{S_1, S_2, ..., S_j\}$ ($\{S_{j+1}, S_{j+2}, ..., S_k\}$) further to the left and thus adding the shift value $c_i \leq 0$ to all D_is. Since c_is are non$-$positive, the maximum shift cannot be reduced.

■

Theorem 5.3.9. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$ with $L_1 + L_2 < 2rn$. There is an $O(n^2 \log n)$ algorithm that solves the MinMax optimization problem of covering two line segments B_1 and B_2 so that the maximal value of the shift of any sensor is minimized.*

Proof. Let n_1 and n_2 be the minimum number of sensors that are needed to completely cover barriers B_1 and B_2, respectively; i.e. $n_1 = \lceil \frac{L_1}{2r} \rceil$ and $n_2 = \lceil \frac{L_2}{2r} \rceil$. We consider two different cases depending on the distance between B_1 and B_2 .

i) $le(B_2) - re(B_1) \geq 2r$. Therefore, no sensor can cover a part of B_1 and a part of B_2 at the same time, and thus the problems of covering B_1 and B_2 are independent of each other. We consider the following possibilities:

   (a) $n_1 + n_2 < n$. Since $L_1 + L_2 < 2rn$, this implies that $n_1 + n_2 = n - 1$. In this case complete coverage is not possible, and we aim at maximizing the coverage while minimizing the maximum movement. This can be done in $O(n^2)$ using the algorithm in Section 6.2.1.

   (b) $n_1 + n_2 \leq n$. Let α be the smallest number in $\{1, 2, ..., n\}$ such that $2r\alpha \geq L_1$ and $(2r - n)\alpha \geq L_2$. Let β be the greatest number in $\{1, 2, ..., n\}$ such that $2r\beta \geq L_1$ and $(2r - n)\beta \geq L_2$. Let $j = \lceil (\alpha + \beta)/2 \rceil$ and solve the two MinMax problems for B_1 with sensor set $N_1 = \{S_1, S_2, ..., S_j\}$ and B_2 with sensor set $N_2 = \{S_{j+1}, S_{j+2}, ..., S_n\}$ with time complexity $O(n^2)$, using the algorithm in [CKK+09] for $R > L$. Let $rs(B_1)$, $ls(B_1)$, $rs(B_2)$, and $ls(B_2)$ be the maximum right shift in N_1, the maximum left shift in N_1, the maximum right shift in N_2, and the maximum left shift in N_2 respectively. Using Lemma 5.3.8, if the maximum shift is a right (left) shift either the cutting point j is an optimal solution, or an optimal solution can be found using another cutting point in N_2 (N_1). Clearly, an optimal solution can be obtained by repeating the same procedure. Since we are cutting our set into half each time, this can be done in $\log n$ steps, and thus finding an optimal solution in $O(n^2 \log n)$ time, which completes the proof.

ii) $le(B_2) - re(B_1) < 2r$. Thus, it is possible that a sensor in the solution for covering

B_1 (B_2), covers a part of B_2 (B_1), and therefore it is possible to obtain complete coverage with a smaller maximum shift by not covering the part of B_2 (B_1) that is already covered by B_1 (B_2). To ensure that this doesn't happen, we solve the problem of complete coverage for the virtual barrier $[le(B_1), re(B_2)]$ as well as covering the barriers separately, and we return the better solution.

■

Corollary 5.3.10. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(\max(n^2(\log n)^k, n^2 + mn))$ algorithm that solves the MinMax optimization problem of covering $m = 2^k$ line segments B_1 and B_2, ..., B_m with $\sum_{i=1}^{m} |B_i| < 2rn$ so that the maximal value of the shift of any sensor is minimized.*

Proof. When complete coverage is not possible, the problem is similar to the case where $\sum_{i=1}^{m} |B_i| = 2rn$, and can be solved with time complexity $O(n^2 + mn)$. Otherwise, the cutting points can be found in $O((\log n)^k)$, resulting in total time complexity $O(\max(n^2(\log n)^k, n^2 + mn))$. ■

5.4 The MinMax Problem on a Circle

We are interested in the problem of protecting an area by detecting intruders as they enter a protected region by passing through a barrier. Such a barrier can be usually modeled by a closed curve. Thus, a circle is a more realistic model to represent the barrier than a straight line. Furthermore, any results that are valid on a circular barrier approximate very well the problem for smooth curves on which each sensor covers the same segment size of the curve.

Let $S = \{S_1, S_2, ..., S_n\}$ be a set of sensors with initial polar positions $x_1 = (\theta_1, r), x_2 = (\theta_2, r), ..., x_n = (\theta_n, r)$ arbitrary dispersed on the circumference of barrier

circle $C = (o, r)$, where r is radius of C and o is the origin. Since all the sensors are on the circumference of the circle, for simplicity, we can refer to their positions by only using their angles. Assume that sensors are sorted by their initial positions, $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$, and that they all have the same coverage c_r. The coverage of any sensor S, c_r, is the length of the arc covered by S on the barrier, as illustrated in Figure 38. Since all the other coordinates are angular, we use the angle of coverage r_Ω instead of c_r. Furthermore, it should be noted that the results of this chapter are not limited to a circular barrier and are still valid for any smooth curve as long as the sensors provide equal coverage.

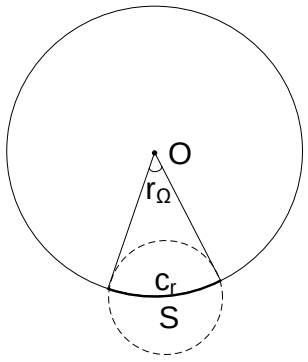
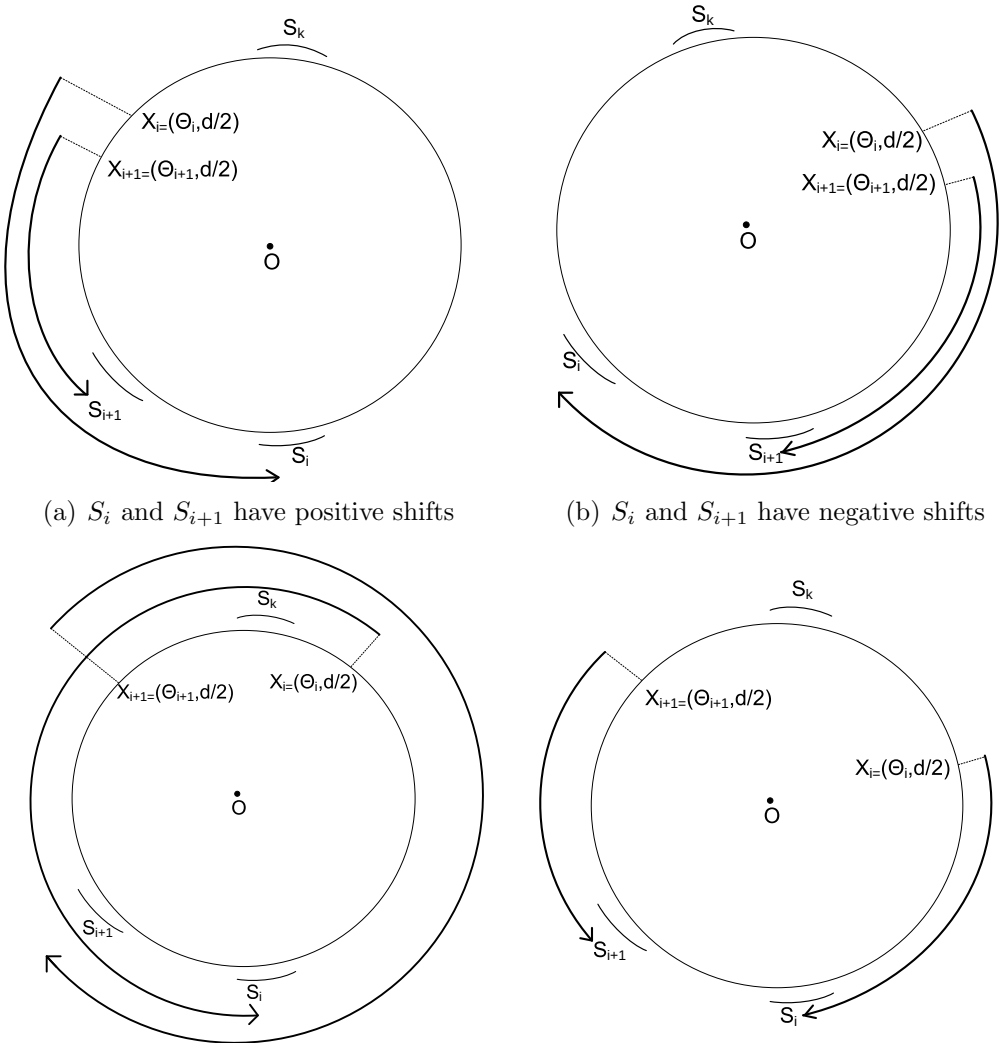


Figure 38: The coverage of sensor S on the barrier $C = (o, r)$

Here, we consider the problem of assigning final positions to sensors in $S = \{S_1, S_2, ..., S_n\}$ so as to provide maximal barrier coverage while minimizing the maximum movement of any sensor along the circle. We refer to this problem as *MinMax* on a circular barrier.

We indicate the dislocation of a sensor S_i by angle α_i, and thus α_i is equal to $r * \min(|\phi_i - \theta_i|, 2\pi - |\phi_i - \theta_i|)$, where θ_i and ϕ_i are the initial and final positions of S_i respectively. If S_i has to travel the distance α_i to reach its final position counterclockwise, the shift of S_i is denoted by α_i, and by $-\alpha_i$, otherwise.

Let L and R be the barrier length ($2r\pi$) and the total coverage of all sensors (nc_r), respectively. We consider all scenarios: $R < L$, $R = L$, and $R < L$. The following



(a) S_i and S_{i+1} have positive shifts

(b) S_i and S_{i+1} have negative shifts

(c) S_i has a positive shift, and S_{i+1} has a negative shift

(d) S_i has a negative shift, and S_{i+1} has a positive shift

Figure 39: Possible scenarios in which S_i succeeds S_{i+1} in the counterclockwise traversal of sensors.

lemma holds for all cases.

Lemma 5.4.1. *Order Preserving Lemma: Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$ on a circular barrier $C = (o, r)$. There exists an optimal solution for the MinMax optimization problem on*

C for which the order of sensors in the counterclockwise traversal of sensors, is the same as the initial order of sensors.

Proof. In order to show that the lemma holds, it is sufficient to show that there exists an optimal solution in which every sensor S_{i+1} immediately follows S_i in the counterclockwise traversal of sensors.

We show that for every optimal solution P, if there are two sensors S_i and S_{i+1}, where S_{i+1} does not immediately follow S_i in the counterclockwise traversal of sensors, it is possible to change the solution to get another optimal solution in which S_{i+1} immediately follows S_i in the counterclockwise traversal of sensors.

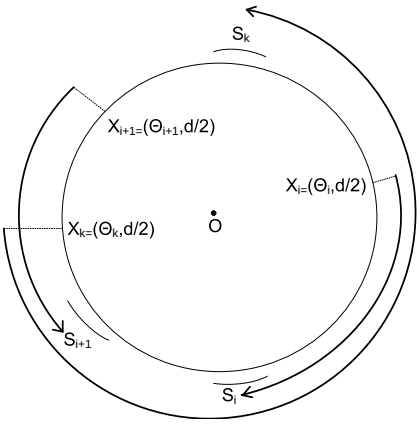
We consider all possible scenarios where S_{i+1} does not immediately follow S_i in the counterclockwise traversal of sensors:

- Both S_i and S_{i+1} have positive shift values [See Figure 39(a)]: Let β be the initial distance between S_i and S_{i+1}; i.e. $\beta = r * (\theta_{i+1} - \theta_i)$. Also, let γ be the distance between the final positions of S_i and S_{i+1}. It is easy to see that $\alpha_i = \alpha_{i+1} + \beta + \gamma$. Swapping S_i and S_{i+1}, would result is shifts $\alpha_{i+1} + \beta$ and $\alpha_{i+1} + \gamma$ which are clearly less than α_i. Thus, swapping S_i and S_{i+1} would cover the same area without increasing the maximal shift.

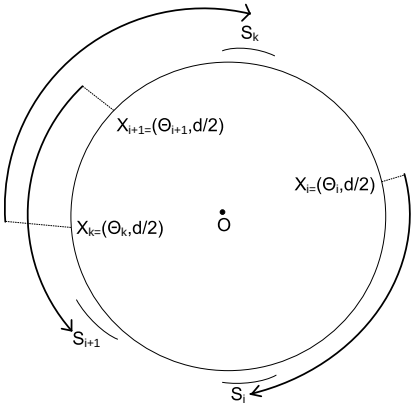
- Both S_i and S_{i+1} have negative shift values [See Figure 39(b)]: Similar to the previous case, we can show that swapping S_i and S_{i+1} would cover the same area without increasing the maximal shift.

- S_i has a positive shift, and S_{i+1} has a negative shift [See Figure 39(c)]: Let β be the distance between final positions of S_i and S_{i+1}. Swapping S_i and S_{i+1} would then result in shifts $\alpha_i - \beta$ and $\alpha_{i+1} - \beta$, and thus covering the same area without increasing the maximal shift.

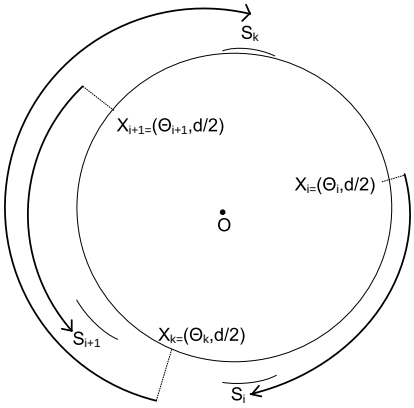
- S_i has a negative shift, and S_{i+1} has a positive shift [See Figure 39(d)]: If in



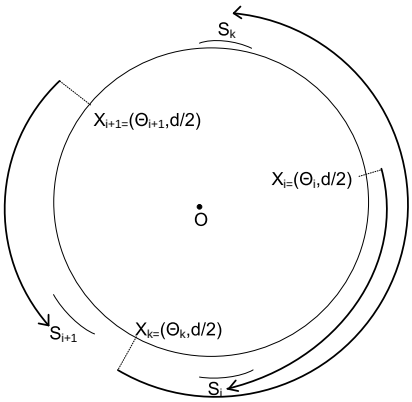
(a) x_k is between x_{i+1} and y_{i+1} and S_k has a positive shift



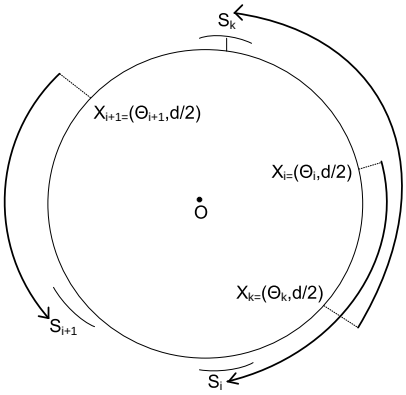
(b) x_k is between x_{i+1} and y_{i+1} and S_k has a negative shift



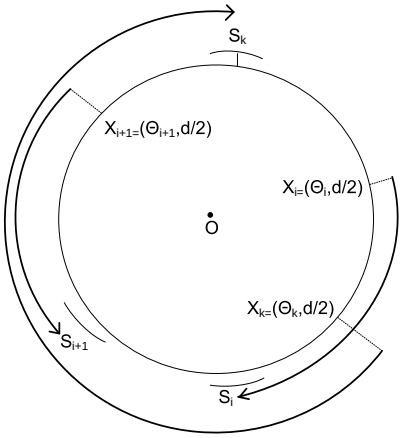
(c) x_k is between y_{i+1} and y_i and S_k has a negative shift



(d) x_k is between y_{i+1} and y_i and S_k has a positive shift



(e) x_k is between y_i and x_i and S_k has a positive shift



(f) x_k is between y_i and x_i and S_k has a negative shift

Figure 40: Possible scenarios in which S_i has a negative shift, and S_{i+1} has a positive shift.

the counterclockwise traversal of sensors, there is no sensor between S_i and S_{i+1}, they are already in order. Therefore, in the counterclockwise traversal of sensors, there should be at least on sensor between S_i and S_{i+1}. Let S_k be such sensor that is the closest to S_{i+1}. One of the following can happen:

1. x_k is between x_{i+1} and y_{i+1}: If S_k has a positive shift value [see Figure 40(a)], swapping S_k and S_i would clearly decrease both α_i and α_k, and thus the new solution provides the same coverage without increasing the maximum shift, while S_i, and S_{i+1} are in order. If S_k has a negative shift value [see Figure 40(b)], swapping S_{i+1} and S_k would decrease both α_{i+1} and α_k by $dist(S_{i+1}, S_i)$. The new solution would have the same coverage, without increasing the maximum shift while the number of sensors between S_i and S_{i+1} has decreased by one.

2. x_k is between y_{i+1} and y_i: If S_k has a negative shift value [see Figure 40(c)], the shift value of S_k, α_k is greater than the shift value of S_{i+1}, α_{i+1}. Therefore, it is sufficient to show that by swapping S_k and S_{i+1} neither of the new shifts exceeds α_k. Swapping S_k and S_{i+1} would result in shift values of $dist(x_k, y_{i+1})$ and $dist(x_{i+1}, y_k)$ which are smaller than $\alpha_k = dist(x_k, y_{i+1}) + \alpha_{i+1} + dist(x_{i+1}, y_k)$. Thus, the new solution has the same coverage without increasing the maximum shift while the number of sensors between S_i and S_{i+1} has decreased by one. On the other hand, if S_k has a positive shift value [see Figure 40(d)], similar to the case where S_k has a negative shift value, we can swap S_i and S_k, and thus the solution would be an optimal solution in which S_i and S_{i+1} are in order.

3. x_k is between y_i and x_i: If S_k has a positive shift value [see Figure 40(e)], similar to the case illustrated in Figure 40(b), we can swap S_i and S_k, and thus obtain an optimal solution in which S_i and S_{i+1} are in order.

Otherwise, if S_k has a negative shift value [see Figure 40(f)], similar to the case depicted in Figure 40(a), swapping S_k and S_{i+1} would result in an optimal solution in which the number of sensors between S_i and S_{i+1} has decreased by one.

Therefore, in any of the steps either we get an optimal solution in which S_i and S_{i+1} are in order, or we decrease the number of sensors between S_i and S_{i+1} which would in at most m steps (m is the number of sensors between S_i and S_{i+1} in the initial optimal solution) give an optimal solution in which S_i and S_{i+1} are in order.

■

Unlike line barriers, where shifting a sensor to the right (left) would monotonically increase its right (left) shift value, on a circular barrier shifting a sensor more than π clockwise (counterclockwise) would change the direction of the shift and the absolute value of shift would eventually decrease. The absolute value of shifts corresponds to a sine wave. Thus, there might be several balance points, where the maximum clockwise and counterclockwise shifts in a connected interval formed by a group of sensors come to equilibrium. For example, assume there are three sensor S_1, S_2 and S_3 with coverage r_Ω, where r_Ω is close to zero. Furthermore, assume that S_1 ,S_2 and S_3 are located at initial positions $\frac{5\pi}{6}$, $\frac{7\pi}{6} + r_\Omega$ and $\frac{11\pi}{6} + 2r_\Omega$ respectively. Moving S_1, S_2 and S_3 to positions 0, r_Ω and $2r_\Omega$ to form a contiguous interval will introduce shift values $-\frac{5\pi}{6}$, $\frac{5\pi}{6}$ and $\frac{\pi}{6}$. Figure 41 depicts different shift values as we shift the interval by $\frac{\pi}{6}$ clockwise at each step, and it shows there are 3 different rotations in which the maximum clockwise and counterclockwise shifts are equal. However, only the middle one is the one that minimizes the maximal shift values. Below, we show how such a point can be obtained.

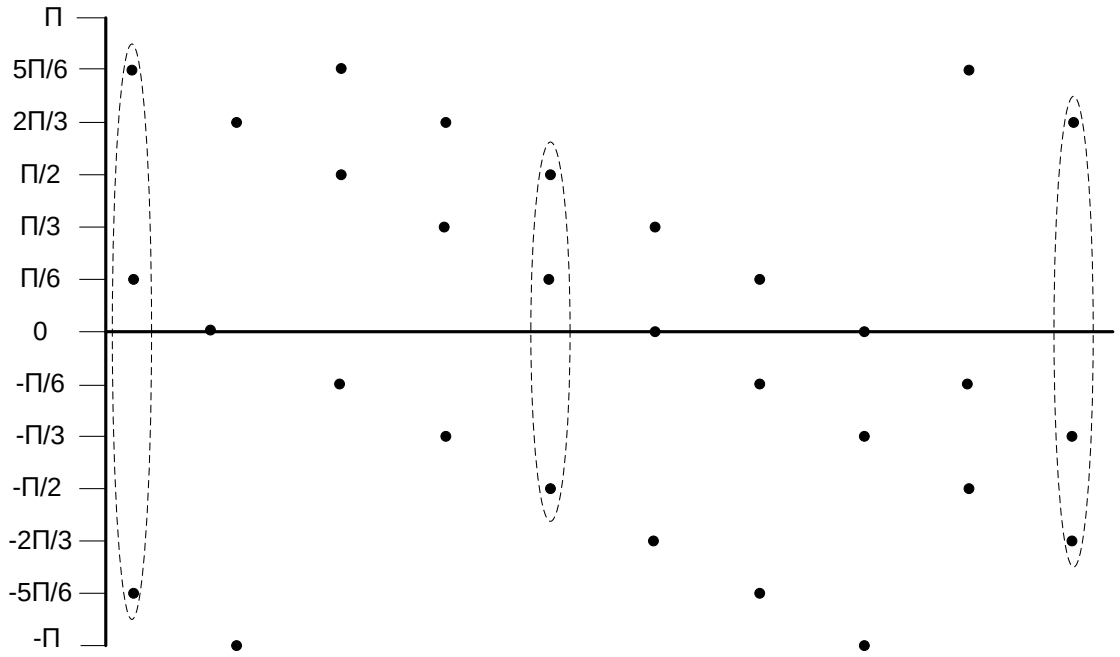


Figure 41: Different shift values of S_1 ,S_2 and S_3 as they are shifted clockwise on C by $\frac{\pi}{6}$ at each step.

Lemma 5.4.2. *Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r initially positioned on a circular barrier $C = (o, r)$. Let P be a solution that leads to an optimal solution for the MinMax optimization problem on C by rotating all sensors in P equally. Furthermore, let $-\pi \leq d_1 \leq d_2 \leq \ldots \leq d_n \leq \pi$ be the sorted shift values of $S_1, S_2, \ldots, S_n$ in P. An optimal solution for the MinMax optimization problem on C can be obtained from P in linear time.*

Proof. Let S_{t_i} be the sensors with shift value d_i, and let δ_i be the modular difference between any two consecutive shift values d_i and $d_{(i+1) \mod n}$.

$$\delta_i = \begin{cases} d_{i+1} - d_i, & i \leq n-1; \\ 2\pi - (d_n - d_1), & i = n. \end{cases}$$

Note that two consecutive shift values do not necessarily correspond to two consecutive sensors. The minimum shift value can be obtained by balancing the maximal clockwise and counterclockwise shifts. However, unlike line barriers, there might be several balance points where the maximal clockwise and counterclockwise shifts are equal on a circle [See Figure 41]. Let δ_j be the maximum value among all δ_is. A minimum balance point can be obtained by shifting the sensors such that S_{t_i} and $S_{t_{i+1}}$ have the maximum clockwise and counterclockwise shifts respectively. This can be easily done by rotating all sensors counterclockwise by $\pi - \frac{\delta_j}{2} - d_{i+1}$. Clearly, finding δ_j as well as all the computations can be done in linear time. ■

Corollary 5.4.3. *Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r initially positioned on a circular barrier $C = (o, r)$. Let P be any solution to the MinMax coverage problem where the maximum clockwise and counterclockwise shifts are equal, and let $-\pi \leq d_1 \leq d_2 \leq \ldots \leq d_n \leq \pi$ be the sorted shift values of $S_1, S_2, \ldots, S_n$ in P. Furthermore, let δ_i be the modular difference between any two consecutive shift values d_i and $d_{(i+1) \mod n}$.*

$$\delta_i = \begin{cases} d_{i+1} - d_i, & i \leq n-1; \\ 2\pi - (d_n - d_1), & i = n. \end{cases}$$

Let δ be the maximum value among δ_is. If $2(\pi - d_n) \geq \delta$, no better solution can be obtained from P by rotating the sensors.

5.4.1 $R \leq L$

Theorem 5.4.4. *Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$ on a circular barrier $C = (o, r)$ with $nc_r = 2r\pi$. There is an $O(n \log n)$ algorithm for the MinMax optimization problem on C.*

Proof. Since $L = R$, all sensors should be in attached positions in the final solution. Using Lemma 5.4.1, there is an optimal solution in which sensors are in their

initial order. Move sensors $S_1, S_2, ..., S_n$ to positions $0, r_\Omega, 2r_\Omega, ..., (n-1)r_\Omega$ to form a contiguous interval. This can be done in linear time. The optimal solution can then be obtained by rotating the sensors once they are in attached position. Let $d_1 \leq d_2 \leq \ldots \leq d_n$ be the sorted shift values of $S_1, S_2, \ldots, S_n$ in this setting. All d_is can be calculated in $O(n \log n)$. Using Lemma 5.4.2, the optimal solution can be obtained by rotation, using an additional $O(n)$ time, which results in total time complexity $O(n \log n)$. ■

When $R < L$, the barrier circumference is greater than the total coverage of all sensors. Thus, complete coverage is not possible. Consequently, we consider two optimization problems to provide maximal coverage, where sensors can either form a contiguous interval or a set of contiguous intervals.

5.4.1.1 Non-contiguous Coverage

Since $R < L$, and all the sensors are already on the barrier, the non-contiguous MinMax optimization problem reduces to the problem of eliminating all the overlaps, while keeping the maximum movement of any sensor minimized.

Theorem 5.4.5. *Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$ on a circular barrier $C = (o, r)$ with $nc_r < 2r\pi$.. There is an optimal $O(n)$ algorithm for the non-contiguous MinMax optimization problem on C.*

Proof. Let S_i be the sensor with the smallest index that intersects with the next sensor S_{i+1}; i.e. $\theta_{i+1} - \theta_i < r_\Omega$. Let $L_i = \{S_t, S_{t+1}, ..., S_i\}$ be the set of sensors with indices less than or equal to i in attached position to S_i. Furthermore, let $ms(L_i)$ be the current maximum shift of L_i in any direction (initially for all i, $L_i = 0$). Let o_i be the overlap between S_i, and S_{i+1}; i.e. $o_i = r_\Omega - \theta_{i+1} + \theta_i$. If $|ms(L_i)| \geq o_i$,

shift S_{i+1}, o_i clockwise. Clearly, this does not change the maximum shift. Otherwise, shift S_{i+1}, $\frac{o_i-|ms|}{2}$ counterclockwise, and shift sensors in L_i by $\frac{o_i-|ms|}{2}$ clockwise. If $L_i' = L_i \cap \{S_1, ..., S_{t-1}\} \neq \emptyset$, shift sensors in L_i' by $\frac{o_i-|ms|}{2}$ clockwise, and let $L_i = L_i' \cup L_i$. Also, update $ms(L_i)$ to be the current maximum shift of L_i. It is obvious that the overlap can not be eliminated with a smaller shift value. The same procedure can be repeated until there are no more overlaps. Furthermore, each step takes constant time and since there are at most n steps, all final positions can be calculated in linear time. Also, since at each step the maximum shift is increased at most by half of the overlap size, the total maximum shift is at most $\sum_{i=1}^{m} \frac{|o_i|}{2}$. Clearly the sum of the overlaps is less than $2\pi - r_\Omega$. Thus, we have $d_n \leq \pi r - \frac{r_\Omega}{2}$, where d_n is the maximum shift. Furthermore, the maximum difference between any two consecutive shift values, δ, is at most as big as the greatest overlap, and therefore we have $\delta \leq r_\Omega$. This implies $2(\pi - d_n) \geq \delta$. Therefore, according to Corollary 5.4.3, the solution cannot be further improved by rotation. ■

5.4.1.2 Contiguous Coverage

According to Lemma 5.4.1, we only need to consider solutions that preserve the original order of sensors. When complete coverage is not feasible, in a contiguous coverage every sensor S_i is attached to its immediate neighbors in the original setting, $S_{(i-1 \mod n)}$ and $S_{(i+1) \mod n}$, except for the two endpoints of the contiguous interval. It is natural to consider all the n possible contiguous intervals $S_j, S_{(j+1) \mod n}, ..., S_{(j+n-1) \mod n}$ and use the algorithm in the proof of Theorem 5.4.4 for each of the n contiguous intervals. The optimal solution can be then gained in $O(n^2 \log n)$ by selecting the contiguous interval which minimizes the maximum shift. However, by using two additional data structures, we can improve the complexity of the algorithm to $O(n \log n)$ as shown below.

Theorem 5.4.6. *Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$ on a circular barrier $C = (o, r)$ with $nc_r < 2r\pi$. There is an $O(n \log n)$ algorithm for the contiguous MinMax optimization problem on C.*

Proof. Using Lemma 5.4.1, there exists an optimal solution in which all sensors are in their initial order. However, since $R < L$, in a contiguous coverage, there would be exactly one non-covered interval on the barrier between the two endpoints of the contiguous interval formed by sensors in attached positions. Let P_i be the problem of covering the barrier with a contiguous interval formed by sensors $S_i, S_{(i+1) \mod n}, ..., S_{(i+n-1) \mod n}$, while the maximum movement of any sensor is minimized. The optimal solution can be then gained by finding a P_i with minimum maximal shift.

We now show how to inductively find the maximal shift in an optimal solution for P_i. We use two data structures to get P_{i+1} from P_i in $O(\log n)$ time. In order to find the maximal shift in an optimal solution for P_1, form a contiguous interval by assigning positions $y_1 = 0, y_2 = r_\Omega, ..., y_n = (n-1)r_\Omega$ to $S_1, S_2, ..., S_n$. This can be done in linear time. Let T be a balanced binary search tree holding the shift values of sensors when $S_1, S_2, \ldots, S_n$ are moved to $0, r_\Omega, \ldots, (n-1)r_\Omega$. Also, let T' be another balanced binary search tree holding the difference between consecutive shift values ($\delta_i = d_{(i+1) \mod n} - d_i$). Using Lemma 5.4.2, and noting that maximum δ_i can be obtained from T' in constant time, the maximal shift in an optimal solution for P_1 can be obtained in constant time.

Similarly, P_{i+1} can be obtained from P_i in constant time, once T and T' are updated. T can be updated in logarithmic time to hold the shift values for P_{i+1} by removing the shift value of S_i when it is at position $(i-1)r_\Omega$ from T and adding the new shift value of P_i to T, when it is moved to position $-(n-i+1)r_\Omega$. Furthermore,

the corresponding δ_i should be updated in T'. This can be done in logarithmic time as well.

Thus, once T and T' are initially constructed in $O(n \log n)$, every step takes $O(\log n)$, resulting in total time complexity $O(n \log n)$.

■

5.4.2 $R > L$

When the coverage of sensors is more than the barrier length, the problem reduces to the problem of covering all the gaps such that the maximum movement of any sensor is minimized.

Lemma 5.4.7. *(Sufficient condition for optimality) Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$ on a circular barrier $C = (o, r)$ with $nc_r > 2\pi r$. Let P be an order preserving solution to the MinMax optimization problem on C. Let $d_1 \leq d_2 \leq \cdots \leq d_n$ be the sorted shift values of sensors in P. If $d_n = 0$, clearly no solution could have a maximum shift less than d_n. For any $0 < d_n \leq \pi - \frac{r_\Omega}{2}$, if the following conditions are satisfied, then no optimal solution to the MinMax problem on C has a maximum shift less than d_n.*

a) There exists a pair of sensors S_i and S_j, such that the counterclockwise shift of S_i is equal to d_n, the clockwise shift of S_j is equal to d_n, and sensors $S_i, S_{(i+1) \mod n}, ..., S_j$ are in attached positions.

b) The difference between any two consecutive shift values is at most r_Ω.

Proof. Let P be a solution of an instance of the MinMax optimization problem on C with $nc_r > 2\pi r$ ($nr_\Omega > 2\pi$) satisfying the condition of the lemma with sensors S_i and S_j as given in condition (a) . Let P' be an optimal solution to the same problem with maximum shift less than d_n. Note that the shift value of sensor S_i in

P is equal to d_n. According to Lemma 5.4.1, we only need to consider solutions that preserve the original order of sensors. Let α_i be the angular position of sensor S_i in solution P. Clearly P' must place S_i in position $\alpha_i - \beta$ where $0 \leq \beta \leq 2d_n$. Since P' is an order preserving solution and $S_i, S_{(i+1) \mod n}, ..., S_j$ are in attached positions, $S_i, S_{(i+1) \mod n}, ..., S_j$ should be moved by at least β, so as provide complete coverage. Since the maximum difference between any two consecutive shift values δ is at most r_Ω, and $d_n \leq \pi - \frac{r_\Omega}{2}$, this implies $2(\pi - d_n) \geq \delta$. According to Corollary 5.4.3, shifting sensors $S_i, S_{(i+1) \mod n}, ..., S_j$ clockwise by β does not lead to a better solution than P, which contradicts the fact that P' has a maximum shift less than d_n. ■

Before giving an algorithm for the MinMax optimization problem on a circular barrier with $nc_r > 2r\pi$, we need to introduce two more definitions. Let o_i be the overlap between sensors S_i and $S_{(i+1) \mod n}$.

$$o_i = \begin{cases} 0, & \text{if } |\theta_{(i+1) \mod n} - \theta_i| \geq r_\Omega; \\ r_\Omega - |\theta_{(i+1) \mod n} - \theta_i|, & \text{otherwise.} \end{cases}$$

Also, we introduce the the term *shift by θ with gap preservation* to be the following.

Definition Let k an integer such that $\sum_{j=0}^{k-1} |o_{(i+j) \mod n}| < \theta \leq \sum_{j=0}^{k} |o_{(i+j) \mod n}|$. Also, let k' be an integer such that $\sum_{j=0}^{k-1} |o_{(i-1-j) \mod n}| < \theta \leq \sum_{j=0}^{k} |o_{(i-1-j) \mod n}|$. We say S_i has been shifted by θ with gap preservation clockwise (counterclockwise) if S_i has been shifted clockwise (counterclockwise) by θ and for every $1 \leq j \leq k$, sensor S_j has been shifted clockwise (counterclockwise) by $\theta - \sum_{l=1}^{j} |o_{(i+l) \mod n}|$ $(\theta - \sum_{l=1}^{j} |o_{(i-l) \mod n}|)$.

Theorem 5.4.8. *Let $S_1, S_2, ..., S_n$ be sensors with sensing coverage c_r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq ... \leq \theta_n \leq 2\pi$ on a circular barrier $C = (o, r)$ with $nc_r > 2\pi r$. There is an $O(n^2)$ algorithm for the MinMax optimization problem on C.*

Proof. According to Lemma 5.4.7, we can obtain the optimal solution by covering all gaps while always satisfying both conditions *(a)* and *(b)*. Let α_{cw} and α_{ccw} be the current maximum clockwise and counterclockwise shift respectively and let $x = max(\alpha_{cw}, \alpha_{ccw})$. Initially $\alpha_{cw} = \alpha_{ccw} = 0$.

Let $g_1, g_2, \ldots, g_l$ be all the gaps on C to be covered in counterclockwise order. Furthermore, let S_{t_i} and $S_{t_{i+1}}$ be the two sensors defining the gap g_i; i.e. $g_i = [\phi_{t_i} + \frac{r_\Omega}{2}, \phi_{t_i+1} - \frac{r_\Omega}{2}]$, where ϕ_j is the current position of S_j in the solution. We specify how to cover the gaps inductively while maintaining as an invariant the conjunction of the conditions (a) or (b) of Lemma 5.4.7. Note that as long as the sensors are shifted by gap preservation only, the difference between sorted shifts can be at most as big as the largest overlap r_Ω. Furthermore, at every step the maximum shift is incremented by at most half of the current gap that is being covered. Consequently the maximum gap is at most $\frac{1}{2}\sum_{i=1}^{l} g_i \leq \frac{1}{2}(2\pi - r_\Omega)$, and thus condition (b) of Lemma 5.4.7 is always satisfied. Therefore, we only need to verify condition (a) is satisfied at every step.

First we show how to cover g_1 satisfying the invariant. Shift S_{t_1} and S_{t_1+1} counterclockwise and clockwise respectively with gap preservation by $\frac{g}{2}$. The maximum shift x is now $\frac{g}{2}$, and S_{t_1} has counterclockwise shift x, S_{t_1+1} has clockwise shift x, and all sensors in between (zero sensors) are in attached positions. Thus, the invariant holds between sensors $inv_e = S_{t_1+1}$ and $inv_b = S_{t_1}$. For brevity, we will say that the condition *(a)* holds at node inv_e.

Assume gaps $g_1, g_2, \ldots g_{i-1}$ are all covered satisfying the conditions (a) and (b), just before we cover gap g_i. After covering g_{i-1}, inv_e has been assigned to be the sensor node with maximum index such that its clockwise shift equals x and such that it is preceded by the node inv_b whose counterclockwise shift equals x, and all intermediate nodes are in attached position. We define $cwsurplus(g_i)$ to be the surplus sensor range

starting counterclockwise at inv_e up to the node S_{t_i} and we define $ccwsurplus(g_i)$ to be the surplus sensor range starting counterclockwise at S_{t_i+1} up to the node inv_b. Since $R > L$, we have $cwsurplus(g_i) + ccwsurplus(g_i) > 0$ and thus there are three possible scenarios depending on the values of $cwsurplus(g_i)$ and $ccwsurplus(g_i)$.

i) Both clockwise and counterclockwise surpluses are greater than zero: in this case neither S_{t_i} nor S_{t_i+1} has been moved and their current shift value is zero. Let $m = min(\frac{g_i}{2}, cwsurplus(g_i), ccwsurplus(g_i), x)$, and shift S_{t_i} and S_{t_i+1} counterclockwise and clockwise respectively with gap preservation by m.

   If $m = \frac{g_i}{2}$, the gap is now closed, and inv_b and inv_e remain the same satisfying both conditions of Lemma 5.4.7. Otherwise, if $m = x$, shift S_{t_i} and S_{t_i+1} counterclockwise and clockwise respectively with gap preservation by $\frac{g}{2} - x$ to cover the gap completely and let $inv_b = S_{t_i}$ and $inv_e = S_{t_i+1}$ satisfying the conditions of Lemma 5.4.7. Otherwise, update the clockwise and counterclockwise surpluses of g_i; $cwsurplus(g_i) = cwsurplus(g_i) - m$ and $ccwsurplus(g_i) = ccwsurplus(g_i) - m$, and either $cwsurplus(g_i) = 0$ or $ccwsurplus(g_i) = 0$.

   - $cwsurplus(g_i) = 0$: This implies that sensor $S_{t_i}, S_{(t_i-1) \bmod n}, \ldots, inv_b$ are all in attached positions, and all the surplus is between S_{t_i+1} up to node inv_b, in the counterclockwise traversal of C. Let $m' = min(x - m, g_i - 2m)$. Shift S_{t_i+1} clockwise with gap preservation by m'. Thus either the gap is covered without changing the invariants or S_{t_i+1} has a clockwise shift equal to x. In the later case, shift both S_{t_i} and S_{t_i+1} counterclockwise and clockwise with gap preservation by half of the remaining gap, $\frac{g_i - 2m - m'}{2}$. Note that since $cwsurplus(g_i) = 0$, $S_{t_i}, S_{(t_i-1) \bmod n}, \ldots, inv_b$ are all in attached positions and the maximum counterclockwise shift occurs at inv_b with value equal to $x + \frac{g_i - 2m - m'}{2}$. Also, since the gap is now closed S_{t_i+1} is now in attached positions with $S_{t_i}, S_{(t_i-1) \bmod n}, \ldots, inv_b$, with the maximum clockwise shift

value equal to $x+\frac{g_i-2m-m'}{2}$. Let S_{t_i+1} be inv_e and update x to be $x+\frac{g_i-2m-m'}{2}$ satisfying the condition of Lemma 5.4.7.

- $ccwsurplus(g_i) = 0$: Thus all the surplus is between inv_e up to the node S_{t_i} int he counterclockwise traversal of the circle. Reverse the order of covering gaps $g_i, g_{i+1}, ..., g_l$, and start by covering g_l. Note that g_l is the first gap after inv_b in the clockwise traversal of the circle and the problem reduces to the case where $cwsurplus(g_i) = 0$. Also, it should be mentioned that this order reversal procedure happens at most once during the course of execution of the algorithm.

ii) The clockwise surplus is zero: Thus all the surplus is between S_{t_i+1} up to node inv_b, in the counterclockwise traversal of C. In this case either both S_{t_i} and S_{t_i+1} have shift value zero or they both have equal clockwise shift b. If the shift values are zero, follow the algorithm for case (i). Otherwise, let $m = min(x - b, g_i)$. Shift S_{t_i+1} by m clockwise with gap preservation. If $m = g_i$, the gap is now covered, and inv_b and inv_e remain the same, satisfying both conditions of Lemma 5.4.7. Otherwise S_{t_i+1} has a clockwise shift equal to x. Close the remaining gap $g_i - m$, by shifting S_{t_i+1} with gap preservation by $\frac{g_i-m}{2}$ clockwise, and shifting S_{t_i} with gap preservation by $\frac{g_i-m}{2}$ counterclockwise. Furthermore, since $S_{t_i}, S_{(t_i-1) \mod n}, \ldots, inv_b$ are all in attached positions, the invariant now holds at S_{t_i+1}, inv_b with maximum shift value $x + \frac{g_i-m}{2}$.

iii) The counterclockwise surplus is zero: Reverse the order of covering gaps $g_i, g_{i+1}, ..., g_l$, and start by covering g_l, using the algorithm for the case where $cwsurplus(g_i) = 0$.

■

5.5 Conclusion

We studied the MinMax optimization problem both on multiple line barriers and circular barriers for all possible scenarios where the barrier (sum of barriers) length is smaller than, equal to or greater than the total coverage of all sensors. When sensors had unequal ranges, we showed that all three optimization problems on a line segment barrier as well as circular barriers were NP-hard. In contrast, when sensors had equal ranges, we presented several efficient algorithms to solve the optimization problems stated above. All our algorithms were centralized: they were given initial positions of sensors and they calculated optimized final positions. A summary of the complexities of the algorithms given for all cases for sensors with identical ranges r, is given in Table 2. Furthermore, since $O(n)$ is a trivial lower bound for the time complexities of all the algorithms presented in this chapter, all our linear algorithms are optimal.

Table 2: Algorithm complexities for the MinMax problem for homogeneous sensors.

	Contiguous	Non-contiguous
m line segments $R = 2rn = \sum_{i=1}^{m} L_i$	$O(n^2 + mn)$	$O(n^2 + mn)$
m line segments $2rn < \sum_{i=1}^{m} L_i$	$O(\max(n(\log n)^{\log m}, n^2 + mn))$	$O(\max(n(\log n)^{\log m}, n^2 + mn))$
m line segments $2rn > \sum_{i=1}^{m} L_i$	$O(\max(n^2(\log n)^{\log m}, n^2 + mn))$	$O(n^2 + mn)$
Circular barrier $c_r n = L = 2\pi r$	$O(n \log n)$	N.A.
Circular barrier $cr_n < L = 2\pi r$	$O(n \log n)$	$O(n)$
Circular barrier $c_r n > L = 2\pi r$	$O(n^2)$	N.A.

Chapter 6

Minimizing the Maximum Number of Sensors Moved for Barrier Coverage

In this chapter, we introduce the problem of achieving maximal coverage of a barrier with movable sensors while minimizing the number of sensors moved *MinNum*. Minimizing the number of sensors moved can minimize the total energy especially when the energy needed to initiate a movement is significantly large. Although minimizing the maximum movement and minimizing the sum of sensor movements have been studied before [BBH+08, CKK+10, CKK+09, SLX+10, DHM+09, WCLP06], to the best of our knowledge, the MinNum problem has never been studied for the barrier coverage problem.

In fact, [DHM+09] is the only study considering the MinNum problem for mobile sensors. The authors in [DHM+09] consider n mobile nodes initially dispersed on the plane with different transmission ranges and they move the sensors so as to achieve a final configuration property on the graph induced by the nodes. They refer to this

problem as the movement problem and they consider three different optimization problems so as to achieve property P: MinMax, MinSum and MinNum where they aim at minimizing the maximum movement, sum of movements, and the number of sensors moved respectively.

They consider movement problems such as collocation and dispersion, where in the former they consider properties such as connectivity, strong connectivity in case of directed graphs, and connectivity between two specific vertices s and t. In the later, the goal is to distribute the nodes in order to guarantee a minimum pairwise separation between nodes, resulting in an independent set of nodes. The authors also consider the perfect matching property, where they move the sensors into nearby pairs so that these pairs can exchange information.

The results for the problems listed above. The results of their study is shown in Table 3.

Table 3: Summary of results in [DHM$^+$09]

	Max	Sum	Num
connectivity	$O(\sqrt{\frac{m}{OPT}})$	$O(\min\{n, m\})$ $\Omega(n^{1-\epsilon})$	$O(m^\epsilon)$ $O(\log n)$
directed connectivity	$O(\epsilon m)$ $\Omega(n^{1-\epsilon})$	open	$O(m^\epsilon)$ $O(\log^2 n)$
s-t connectivity	$O(\sqrt{\frac{m}{OPT}})$	$O(n)$	polynomial
independence	$\frac{1}{\sqrt{3}}$ additive in $\mathbb{R}^2$	open	PTAS in $\mathbb{R}^2$
perfect matchability	polynomial	polynomial	polynomial

We study the case when the barrier of the region that must be protected is one dimensional, and sensors are initially dispersed arbitrarily on the infinite line containing the barrier. It should be mentioned that the discrete case where sensors can only move to specific positions can be solved similarly to the case where the barrier length is equal to the total coverage of sensors. We consider single line segment and multiple

line segment barriers as well as circular barriers. All our algorithms presented in this chapter are centralized polynomial algorithms. Furthermore, we show that even for a single line segment barrier, the problem remains challenging and we show it to be NP-hard for some cases.

The remainder of this chapter is organized as follows. Section 6.1 shows NP-hardness results for sensors with unequal ranges on an infinite line as well as a line segment, and presents efficient algorithms for identical range sensors. Section 6.2 deals with sensors with identical ranges on multiple barriers. Finally, Section 6.3 considers the MinNum problem on a circular barrier and presents polynomial algorithms for identical range sensors as well as NP-hardness results for arbitrary range sensors.

6.1 The MinNum Problem on a Single Line Barrier

We consider the problem of minimizing the number of sensors moved so as to achieve maximal coverage on a line barrier $I = [0, L]$ as well as an infinite line. We first show that when sensors have arbitrary ranges the problem is NP-hard for most cases. Then, we consider the problem with sensors with identical ranges.

6.1.1 Definitions and Preliminaries

We assume that a *barrier* is a closed interval $I = [0, L]$ on the real line. Furthermore, we define the set $S = \{S_1, S_2, ..., S_n\}$ to be the set of sensors dispersed arbitrarily on the real line with initial positions $x_1 \leq x_2 \leq ... \leq x_n$ with sensing ranges $r_1, r_2, ..., r_n$. Thus, a sensor S_i covers the closed interval $C(S_i) = [x_i - r_i, x_i + r_i]$ of length $2r_i$. The sum of the coverage lengths of all sensors $\sum_{i=1}^{n} 2r_i$ is denoted by R. The barrier coverage of a set S, $C(S)$, is the union of the intervals covered by its sensors on

the barrier I; i.e. $C(S) = \bigcup_{S_i \in S} C(S_i) \cap [0, L]$. We are interested in the problem of minimizing the number of sensors that must be moved in order to achieve barrier coverage of $[0, L]$; i.e. $C(S) = [0, L]$. Throughout this study, we refer to this problem as *MinNum optimization problem*. Clearly the MinNum optimization problem is only feasible if $R \geq L$. For the case where $R < L$, we are interested in minimizing the number of sensors that must be moved so as to cover either a sub-interval of length R or sub-intervals of total length R. We refer to the former as the *contiguous MinNum optimization problem*, and to the latter as the *non-contiguous MinNum optimization problem*.

6.1.2 Unequal Range Sensors

In this section, we consider sensors with unequal ranges, and we discuss all possible scenarios. In every case, we either give polynomial algorithms or we show that the problem is NP-hard. We first consider the maximal MinNum optimization problem on an infinite line, and we present an algorithm for the non-contiguous MinNum optimization problem. We then present the NP-hardness results.

6.1.2.1 The MinNum Problem on an Infinite Line

When the barrier is an infinite line, the maximal coverage could be either contiguous or non-contiguous.

For the case of non-contiguous coverage, we can use the greedy algorithm for the activity selection problem [CLRS01] defined as follows:

Definition *Activity selection:* Given a set S of n activities a_i with start time s_i and finish time f_i, find the maximum size set of mutually compatible activities. Activities i and j are compatible if the half-open internal $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap, that is, i and j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$.

Algorithm 9 Greedy Algorithm for the Activity Selection Problem

Input: A set of activities $a_1, a_2, \ldots, a_n$ sorted by their finish time; i.e. $f_1 \leq f_2 \leq \cdots \leq f_n$
$A \leftarrow \{1\}$
$j \leftarrow 1$
for $i \leftarrow 2$ to n **do**
  if $S_i \geq f_j$ **then**
    $A \leftarrow A \cup \{i\}$
    $j \leftarrow i$
  end if
end for
Return A

Using Algorithm 9, we present below an $O(n \log n)$ algorithm for the non-contiguous MinNum optimization problem.

Theorem 6.1.1. *Let $S_1, S_2, ..., S_n$ be sensors with sensing ranges $r_1, r_2, ..., r_n$ in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(n \log n)$ algorithm that solves the non-contiguous MinNum optimization problem on an infinite line.*

Proof. Maximum coverage on the infinite line would be obtained by eliminating all the overlaps between sensors. This can be achieved in two steps: First find a maximal set of non-intersecting sensors as fixed sensors. Then, assign final positions to the remaining sensors. The former translates to the activity selection problem [CLRS01] of finding the maximum set of non-intersecting activities, where the left endpoint of any sensor S_i represents s_i, the start time of activity a_i and the right endpoint represents the finish time f_i. Once the sensors are sorted by their right endpoints, this can be calculated in linear time. However, since we assume that sensors are sorted by their initial positions, and they may have unequal ranges, first we have to sort them by their right endpoints, resulting in a time complexity of $O(n \log n)$ to find a maximal set of non-intersecting sensors M. Let S_l be the leftmost sensor in M; i.e. $S_l = S_1$. For every sensor S_i in S, if S_i is already in M, S_i stays in its initial position. Otherwise assign final position $y_i = x_l - r_l - r_i$ to S_i, and update the leftmost sensor,

i.e. $S_l = S_i$. This can be achieved with time complexity $O(n)$, resulting in total time complexity of $O(n \log n)$. ■

The next theorem shows that the additional requirement of contiguous coverage results in NP-Completeness.

Theorem 6.1.2. *Let $S_1, S_2, ..., S_n$ be sensors with ranges $r_1, r_2, ..., r_n$ in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. The contiguous MinNum optimization problem on an infinite line when sensors have unequal ranges is NP-hard.*

Proof. We reduce the *partition problem* [GJ90] into the problem of maximal contiguous coverage on an infinite line. The partition problem is defined as follows:

Given a sequence of integers $a_1 \leq a_2 \leq ... \leq a_n$, determine whether there exists a set of indices J such that $\sum_{i \in J} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$.

Given an instance of the partition problem, we transform it to the contiguous MinNum problem on an infinite line for the sensor set $S = \{S_1, S_2, ..., S_n, S_{n+1}, S_{n+2}\}$. The sensors $S_1, S_2, ..., S_n$ have sensing ranges $\frac{a_1}{2} \leq \frac{a_2}{2} \leq ... \leq \frac{a_n}{2}$ and have initial positions 0. Furthermore, sensors S_{n+1}, S_{n+2} with sensing range $\frac{1}{2}$ are at initial positions $2C + \frac{a_n+1}{2}$ and $3C + \frac{a_n+3}{2}$ (see Figure 42), where $C = \frac{1}{2} \sum_{i=1}^{n} a_i$.

Clearly, to achieve contiguous coverage, at least n sensors have to move, and at most two sensors can stay fixed. If there is a set of indices J, such that $\sum_{i \in J} a_i = C$, there is a solution to the contiguous MinNum problem such that the sensors S_{n+1} and S_{n+2} stay in their initial positions and the sensors S_is where $i \in J$ cover the interval between S_{n+1} and S_{n+2}. All the other sensors with indices in $\{1, 2, ..., n\} - J$ cover the interval $[C + \frac{a_n}{2}, 2C + \frac{a_n}{2}]$. The number of sensors moved is n.

If such a partition does not exist, either S_{n+1} or S_{n+2} has a final position different from its initial position and the minimum number of sensors moved is $n + 1$.

Thus, if there is an algorithm that solves the contiguous MinNum optimization problem, we can determine whether the partition problem has a solution. Clearly, the

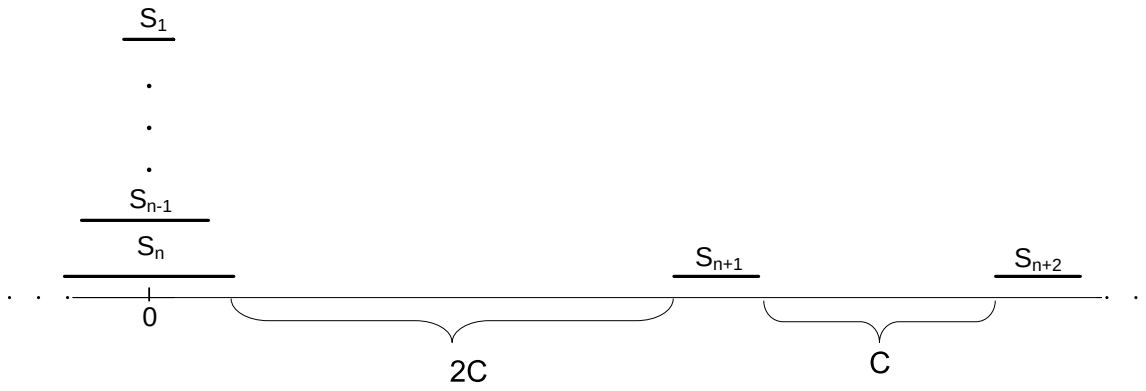


Figure 42: Arrangement of sensors for proving the NP-completeness of the contiguous MinNum problem on an infinite line.

transformation from the partition problem to the contiguous MinNum optimization problem is polynomial. ■

6.1.2.2 The MinNum Problem on a Line Barrier $I = [0, L]$

Theorem 6.1.3. *Let $S_1, S_2, ..., S_n$ be sensors with sensing ranges $r_1, r_2, ..., r_n$ in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. The MinNum optimization problem on a line segment $I = [0, L]$, where $L = \sum_{i=1}^{n} 2r_i$ is NP-hard, when sensors have unequal sensing ranges.*

Proof. We prove it by reducing the *partition problem* [GJ90] into the MinNum optimization problem. Let $a_1 \leq a_2 \leq ... \leq a_n$ be integers and let $C = \frac{1}{2}\sum_{i=1}^{n} a_i$. Given an instance of the partition problem, we transform it into the MinNum optimization problem on a line segment $I = [0, L]$ with the sensor set $S = \{S_1, S_2, ..., S_{n+1}\}$, where $L = 2C + 1$. For every $1 \leq i \leq n$, S_i with sensing range $\frac{a_i}{2}$ is initially located at position $x_i = -\frac{a_n}{2}$. Also, there is a sensor S_{n+1}, with sensing range $\frac{1}{2}$ initially at position $C + \frac{1}{2}$ (see Figure 43). If there is a set of indices J, such that $\sum_{i \in J} a_i = C$, there is a solution to the MinNum optimization problem such that the sensors cover the segment $I = [0, L]$ and the maximum number of sensors moved is n. Assign final

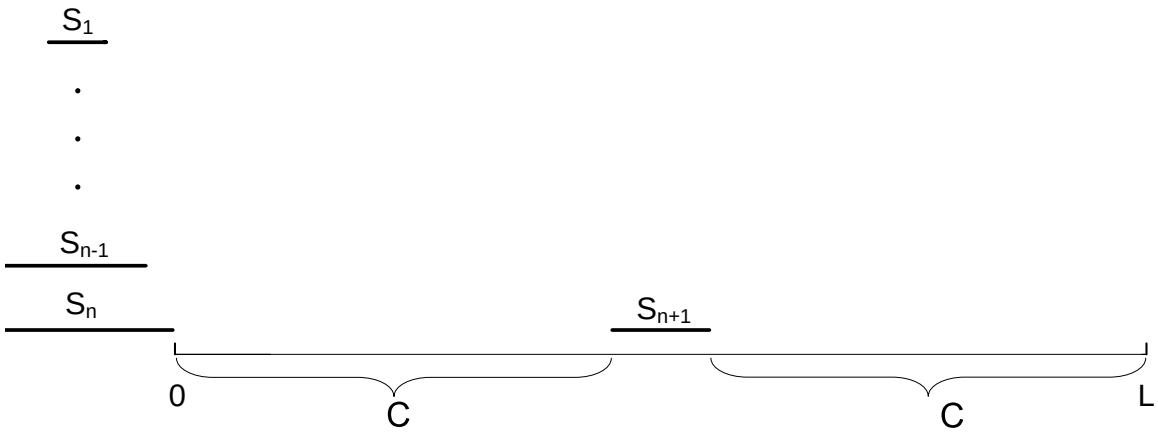


Figure 43: Arrangement of sensors for proving the NP-completeness of the MinNum problem for unequal sensor ranges on a line segment $[0, L]$ where $L = \sum_{i=1}^{n+1} 2r_i$.

positions to sensors with indices in J such that they cover the interval $[0, C]$ and with indices not in J to cover the interval $[C + 1, L]$ and S_{n+1} does not move.

If such a partition does not exist, S_{n+1} has to move as well and thus all sensors move.

Thus, if there is an algorithm that can solve the MinNum optimization problem on a line segment $I = [0, L]$, where $R = L$, we can determine whether the partition problem has a solution. Clearly, the transformation from the partition problem to the MinNum optimization problem is polynomial. ■

It is easy to show that the problem for $R < L$ and $R > L$ remains NP-hard, by using the same proof above and considering a line segment of length $R + \epsilon$ and $R - \epsilon$ respectively, where ϵ is less than the minimum range of sensors. Observe that the argument holds for the non-contiguous coverage as well as the contiguous coverage for the case $R < L$.

6.1.3 Equal Range Sensors

In view of the NP-completeness results, we consider sensors with equal sensing ranges. In contrast to unequal sensor ranges, we present efficient algorithms for all subcases. The rest of this section is organized as follows. Section 6.1.3.1, provides algorithms for the contiguous as well as for the non-contiguous coverage when the barrier is an infinite line. Sections 6.1.3.2, 6.1.3.3, and 6.1.3.4, give algorithms for the coverage of a line segment $I = [0, L]$ for cases where $R = L$, $R < L$ and $R > L$ respectively.

6.1.3.1 The MinNum Problem on an Infinite Line Barrier

Since the barrier is an infinite line, we are looking for maximal coverage, and we consider both contiguous and non-contiguous coverage.

For contiguous coverage, we present an $O(n^2)$ algorithm.

Theorem 6.1.4. *Let $S_1, S_2, ..., S_n$ be sensors with identical range r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(n^2)$ algorithm that solves the contiguous MinNum optimization problem on the infinite line.*

Proof. Since the final positions of sensors introduce a contiguous interval, the final positions would be equally distanced, where every two consecutive sensors are distance $2r$ apart. Let M_i be a maximal set of sensor nodes succeeding S_i with distinct initial positions and at distances which are multiples of $2r$ from S_i that can form a contiguous interval; $M_i = \{S_k | (x_i - x_k) \mod 2r = 0 \wedge x_k - x_i \leq 2r(n-1) \wedge \forall S_k \forall S_m (S_k \in M_i \wedge S_m \in M_i) \rightarrow x_k \neq x_m\}$. Clearly, for every i, M_i can be calculated in $O(n)$ time.

Let M_m be a set among M_is with maximum cardinality, which can be found in $O(n^2)$. Assume that there is an optimal solution with more than $|M_m|$ fixed sensors. Let S_{min} be the leftmost sensor in the optimal solution that has not moved. The maximum number of fixed sensors in the optimal solution would then be equal to

$|M_{min}|$ which contradicts the fact that the optimal solution has more than $|M_m|$ fixed nodes.

To fill the gaps between sensors in M_m, for every $0 \leq i \leq n-1$, if there is no sensor with initial position $x_m + 2ri$, assign final position $y_j = x_m + 2ri$ to a sensor $S_j \in S - M_m$, and remove S_j from S. This provides a contiguous interval, since $|x_i - x_m| \leq 2r(n-1)$ for every S_i in M_m. Clearly the algorithm takes time $O(n^2)$. ■

For the non-contiguous MinNum optimization problem, since sensors have equal ranges, they are already sorted by their right endpoints, and thus we can use the linear algorithm presented in Theorem 6.1.1 on the set of sensors that are completely contained in $I = [0, L]$ to find a maximal set of non-intersecting sensors.

Corollary 6.1.5. *Let $S_1, S_2, ..., S_n$ be sensors with identical range r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an optimal $O(n)$ algorithm that solves the non-contiguous MinNum optimization problem on the infinite line.*

6.1.3.2 $L = 2rn$

Theorem 6.1.6. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$, and let $L = 2rn$. There is an optimal $O(n)$ algorithm that solves the MinNum optimization problem on a line segment $I = [0, L]$.*

Proof. Since $L = 2rn$, the final positions of sensors are predetermined; i.e. sensors should be located at positions $Y = \{-r + 2ri | 1 \leq i \leq n\}$. Let M and X_m be the set of sensors at distinct initial positions in Y, and the initial positions of sensors in M, respectively; i.e. $M = \{S_i | x_i \in Y \wedge \forall S_i \forall S_j (S_i \in M \wedge S_j \in M) \rightarrow x_i \neq x_j\}$, $X_m = \{x_i | S_i \in M\}$. Let $N = S - M$ be the set of sensors that are not in M. Also, let R be the set of coordinates to be assigned to the remaining sensors so as to provide contiguous coverage; i.e. $R = Y - X_m$. Assign a final position to every sensor in N by an on-to map from R. Clearly, this can be done in linear time. ■

6.1.3.3 $L > 2rn$

When $L > 2rn$, complete coverage is not possible. Therefore, we study the maximal MinNum optimization problem in both contiguous and non-contiguous scenarios, and we present efficient algorithms for each case.

For the contiguous optimization problem, we give the following quadratic algorithm.

Theorem 6.1.7. *Let $S_1, S_2, ..., S_n$ be sensors with identical range r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$, and let $L > 2rn$. There is an $O(n^2)$ algorithm that solves the contiguous MinNum optimization problem on a line segment $I = [0, L]$.*

Proof. In order to provide maximum coverage all sensor nodes should be used, and the final positions should be in $B = [r, L-r]$. Let $C = \{S_i \in S | r \leq x_i \leq L-r\}$. The final positions would be equally distanced on B, and every two consecutive sensors would be distance $2r$ apart. For every $S_i \in C$, if there is no contiguous interval including S_i, such that the interval is completely contained in $I = [0, L]$ ($\lfloor \frac{x_i - r}{2r} \rfloor + \lfloor \frac{L - x_i - r}{2r} \rfloor < n-1$), let $M_i = \emptyset$. Otherwise, let M_i be a maximal set of sensor nodes S_k in C with distinct initial positions with the following properties:

- The distance between S_k and S_i is a multiple of $2r$.
- It is possible to form a connected interval with S_k and S_i both contained in it; i.e. $|x_i - x_k| \leq 2r(n-1)$.
- $k \geq i$.

For every i, M_i can be calculated in linear time, and thus a set with maximum cardinality, M_m, among all M_is can be found in $O(n^2)$. Let k be the maximum index in M_m. For every $0 \leq i \leq k-1$, if there is no sensor at position $x_m + 2ri$, move a sensor from $S - M_m$ to $x_m + 2ri$ to fill in the gaps. Move the remaining sensors to the

right(left) endpoint of the contiguous interval to form a bigger contiguous interval. The algorithm takes time $O(n^2)$. ■

Although the problem of maximal non-contiguous coverage on $I = [0, L]$ is similar to the problem of maximal non-contiguous coverage on the infinite line, in that the sensor nodes should be moved to eliminate the overlaps, when the barrier is not big enough, it is possible that the gaps between the sensors in a maximal set of non-intersecting sensors are not big enough to fit all the remaining sensors. Thus, different algorithms are needed to find the minimum number of sensors that need to be moved to maximize the coverage depending on the barrier size. We consider two different scenarios: (i) when the barrier length is at least twice the coverage provided by all nodes; i.e. $L \geq 4rn$ and (ii) when the barrier length is smaller than twice the coverage provided by all nodes; i.e. $L < 4rn$.

6.1.3.3.1 $L \geq 4rn$

Lemma 6.1.8. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Let set M be any set of non-intersecting sensors completely contained in $I = [0, L]$; $M \subseteq S = \{S_i | r \leq x_i \leq L - r\}$. If $L \geq 4rn$, there is a solution with $|M|$ fixed sensors that solves the problem of non-contiguous maximal coverage on $I = [0, L]$.*

Proof. Let $M = \{S'_1, S'_2, ..., S'_k\}$ where $S'_i = S_j$ for some $1 \leq j \leq n$. Let S'_0 and S'_{n+1} be two virtual sensors with sensing range 0 at the beginning and end of the barrier respectively; i.e. $re(S'_0) = le(S'_0) = 0$, $re(S'_{n+1}) = le(S'_{n+1}) = L$. Also, let g_i be the gap between S'_i and S'_{i+1}, i.e. $g_i = le(S'_{i+1}) - re(S'_i)$. Hence, $L = 2rk + \sum_{i=0}^{k} g_i$.

Let ϵ_i be $g_i - 2r\lfloor \frac{g_i}{2r} \rfloor$ ($\epsilon_i < 2r$). In order for the solution to be valid on the line segment B, the $n - k$ remaining sensors should fit in the gaps g_is; i.e. it is sufficient to show that $\sum_{i=0}^{k} \lfloor \frac{g_i}{2r} \rfloor \geq n - k$.

Since $L \geq 4rn$, we have,

$$L \geq 2rn + 2rn \geq 2rn + 2rk.$$

Hence,

$$L = 2rk + \sum_{i=0}^{k} g_i = 2rk + 2r \sum_{i=0}^{k} \lfloor \frac{g_i}{2r} \rfloor + \sum_{i=0}^{k} \epsilon_i \geq 2rn + 2rk.$$

Therefore,

$$2r \sum_{i=0}^{k} \lfloor \frac{g_i}{2r} \rfloor \geq 2rn - \sum_{i=0}^{k} \epsilon_i > 2rn - 2rk - 2r.$$

Consequently,

$$\sum_{i=0}^{k} \lfloor \frac{g_i}{2r} \rfloor \geq n - k.$$

which completes the proof. ■

Theorem 6.1.9. *Let $S_1, S_2, ..., S_n$ be sensors with identical range r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. If $L \geq 4rn$, there is an $O(n)$ algorithm that solves the non-contiguous MinNum optimization problem on a line segment $I = [0, L]$.*

Proof. Since $L > 4rn$, all sensors should be used to provide maximum coverage. Therefore, the candidate sensors that can stay in their initial position are the sensors S_i that are completely contained on the barrier; i.e. $r \leq x_i \leq L - r$. The maximum number of non-intersecting sensors can be calculated in $O(n)$ time similar to the problem on the infinite line. Using Lemma 6.1.8, the remaining sensors can fit in the gaps in between the fixed sensors without intersecting, and thus providing maximum coverage with minimum sensor movement. ■

6.1.3.3.2 $2rn < L < 4rn$ When the barrier length is less than twice the maximum coverage provided by all sensors, the greedy algorithm in Theorem 6.1.1 does not necessarily work. Consider the following example where S_1, S_2, S_3 and S_4 are sensors

with initial positions $1, 1.9, 2.8$ and 4.1 respectively, with identical sensing range 0.5 as illustrated in Figure 44. Here, the maximal set of non-intersecting sensors $\{S_1, S_3, S_4\}$ does not solve the non-contiguous MinNum optimization problem, since no sensors can fit in the gaps between $\{S_1, S_3, S_4\}$. The optimal number of fixed sensors in this example is 2. Indeed, we need to find the maximum number of non-intersecting

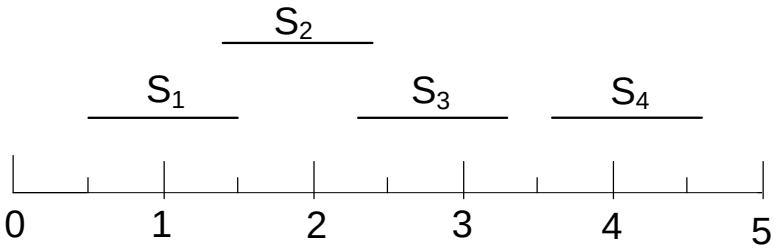


Figure 44: Arrangement of sensors on a line segment $I = [0, 5]$.

sensors k such that remaining $n - k$ sensors can fit in the gaps between the fixed sensors. The problem can be reduced to the problem of finding the maximum-hop path of a certain minimum weight on an edge-weighted graph which represents the sensors, where every vertex on the path represents a sensor that remains fixed in the final position. Since $R < L$, in order to achieve maximal coverage, all sensors need to participate in the coverage. Thus only sensors that are completely included on the barrier can stay fixed and consequently, the vertices of the graph are sensors that are completely contained in the barrier. Furthermore, we need two more sensors to indicate the beginning and the end of the interval. If two sensors S_i and S_j $(i < j)$ do not intersect there is a directed edge between S_i and S_j and the weight of the edge indicates the maximum number of sensors that fit in the gap between S_i and S_j plus the one sensor S_j that has been fixed already and cannot be used toward covering the remaining gaps.

Let S' be the set of sensors that are completely contained in $[0, L]$ and let m be

the size of S'; i.e. $S' = \{S_i \in S | r \leq x_i \leq L - r\}$, $m = |S'|$. We model the sensors with a directed acyclic graph $G = (V, E)$, where $V = S' \cup \{S_0, S_{m+1}\}$, where S_0 and S_{m+1} are virtual sensors at positions 0 and L with sensing ranges 0 indicating the two endpoints of the barrier, and $E = \{(S_i, S_j) | S_i \in S' \wedge S_j \in S' \wedge i < j \wedge x_j - x_i \geq 2r\}$. Also, we define the weight of an edge $w(S_i, S_j)$ as the following:

$$w(S_i, S_j) = \begin{cases} \lfloor \frac{le(S_j) - re(S_i)}{2r} \rfloor + 1 & \text{if } j \leq m, \\ \lfloor \frac{le(S_j) - re(S_i)}{2r} \rfloor & \text{if } j = m + 1. \end{cases}$$

Figure 45 shows the graph representing the sensor arrangement in Figure 44. One can see that there is no 4-hop path with total weight at least 4. The maximum-hop path with weight at least 4, has three hops. The paths $S_0S_1S_4S_5$, $S_0S_2S_4S_5$, and $S_0S_3S_4S_5$ all have weight 4, and they are all optimal solutions with two fixed sensors.

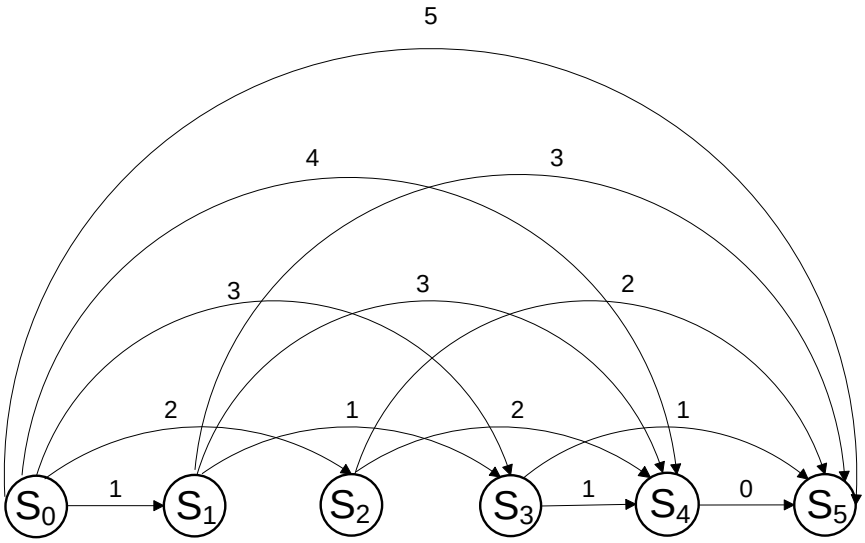


Figure 45: The graph representing sensors arrangement in Figure 44

Theorem 6.1.10. *Let $S_1, S_2, ..., S_n$ be sensors with identical range r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. If $2rn < L < 4rn$, there is an $O(n^3)$ algorithm that solves*

the non-contiguous MinNum optimization problem on a line segment $I = [0, L]$.

Proof. Let the weight of a path P to be equal to the summation of the weights of its edges; i.e. $w(P) = \sum_{e \in P} w(e)$. The number of hops on a path on the graph corresponds to the number of fixed sensors and its weight corresponds to the total number of non-intersecting sensors that can fit on the barrier once the vertices on the path have been fixed. The problem now reduces to finding the maximum-hop path P on G such that $w(P) \geq n$. Let $L(j, k)$ be the weight of a k-hop path P between nodes S_0 and S_j such that $w(P)$ is maximized. A solution $L(n + 1, k)$ is feasible if $L(n + 1, k) \geq n$.

$$L(j,k) = \begin{cases} \max_{0<i<j}(L(i,k-1) + w(S_i, S_j)) & \text{if } k > 1, \\ w(S_0, S_j) & \text{if } k = 1. \end{cases}$$

Thus, using dynamic programming, by calculating $L(j, k)$ for all values of j, k, an optimal feasible solution can be calculated with time complexity $O(n^3)$. Furthermore, in order to calculate the path, we can store the vertex S_i for which $L(i, k-1)+w(S_i, S_j)$ gets its maximum value during the course of calculation of $L(j, k)$. The vertices on the path would then represent the sensors that remain in their initial position. We can assign final positions to the remaining sensors in $O(n)$ time by scanning the barrier from the beginning and filling the gaps with the remaining sensors. ■

6.1.3.4 $L < 2rn$

Since $L < 2rn$, complete coverage is possible. We have the following theorem.

Theorem 6.1.11. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. If $L < 2rn$, there is an $O(n^3)$ algorithm that solves the MinNum optimization problem on a line segment $I = [0, L]$.*

Proof. Let $g_{i,j}$ be the minimum number of sensors that are needed to cover the gap between two sensors S_i and S_j, where $j > i$. In other words:

$$g_{i,j} = \begin{cases} 0 & \text{if } x_j \leq r \vee x_i \geq L - r, \\ \lceil \frac{le(S_j) - 0}{2r} \rceil & \text{if } x_i < -r \wedge x_j \leq L - r, \\ \lceil \frac{le(S_j) - re(S_i)}{2r} \rceil & \text{if } x_i \geq -r \wedge x_j \leq L + r, \\ \lceil \frac{L - re(S_i)}{2r} \rceil & \text{if } x_i \geq r \wedge x_j > L + r, \\ \lceil \frac{L - 0}{2r} \rceil & \text{if } x_i < -r \wedge x_j > L + r. \end{cases}$$

An optimal solution contains a maximal set of sensors S' of size k, where $\sum_{i \text{ and } j \text{ are consecutive sensors in } S'} g_{i,j} \leq (n - k)2r$.

As before, we model the sensor arrangement by a graph G, where nodes represent the sensors and any edge between two sensors S_i and S_j has a weight equal to the minimum number of sensors needed to cover the gap between them plus the one sensor S_j that is already used.

$$w(S_i, S_j) = \begin{cases} g_{i,j} + 1 & \text{if } j \leq n, \\ g_{i,j} & \text{if } j = n + 1. \end{cases}$$

Assume sensors S_1, S_2, S_3, S_4, and S_5, initially positioned at locations $-1, 0.3, 1, 2.7$, and 3.3 with sensing range 0.5 are used to cover the line segment $I = [0, 3]$ (see Figure 46). The graph representing sensors arrangement is illustrated in Figure 47. Sensors S_0 and S_6 represent the two endpoints of the line segment and any path between S_0 and S_6 represents a configuration in which vertices on the path stay fixed. For example, the path $P = S_0S_1S_2S_4S_5S_6$ represent a setting in which sensors S_1, S_2, S_4 and S_5 remain stationary. The weight of the path represents the number of fixed

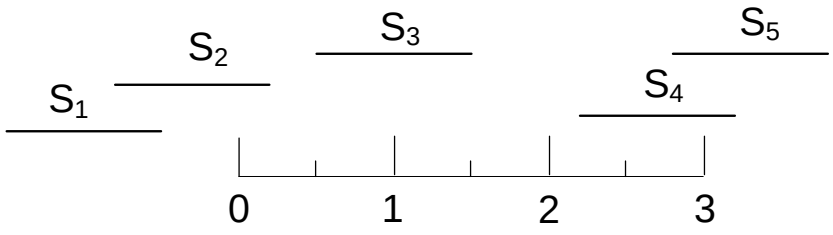


Figure 46: Arrangements of sensors for the coverage of the line segment $I = [0, 3]$

sensors plus the minimum number of sensors that are needed to cover the gaps. In this example P has weight 6, which implies that for such a configuration at least six sensors are needed. However, we only have five sensors, and thus this setting is infeasible.

In general, a path P in G represents a solution in which the sensors on P remain fixed, and the weight of P represents the minimum total number of sensors that are needed to provide complete coverage, once the sensors on P have been fixed. Moreover, we add two virtual sensors S_0 and S_{n+1} with sensing ranges r to represent the beginning and end of the barrier; i.e. $x_0 = -r$, $x_{n+1} = L + r$. An optimal solution would then be a maximum hop path P between S_0 and S_{n+1} such that there are enough sensors to cover the gaps; i.e. $w(P) \leq n$.

Let $L'(j, k)$ be the weight of minimum weight path P between nodes S_0 and S_j that has exactly k hops. $L'(n + 1, k)$ is a feasible solution if $L'(n + 1, k) \leq n$. s.

$$L'(j,k) = \begin{cases} \min_{0<i<j}(L'(i,k-1) + w(S_i, S_j)) & \text{if } k > 1, \\ w(S_0, S_j) & \text{if } k = 1. \end{cases}$$

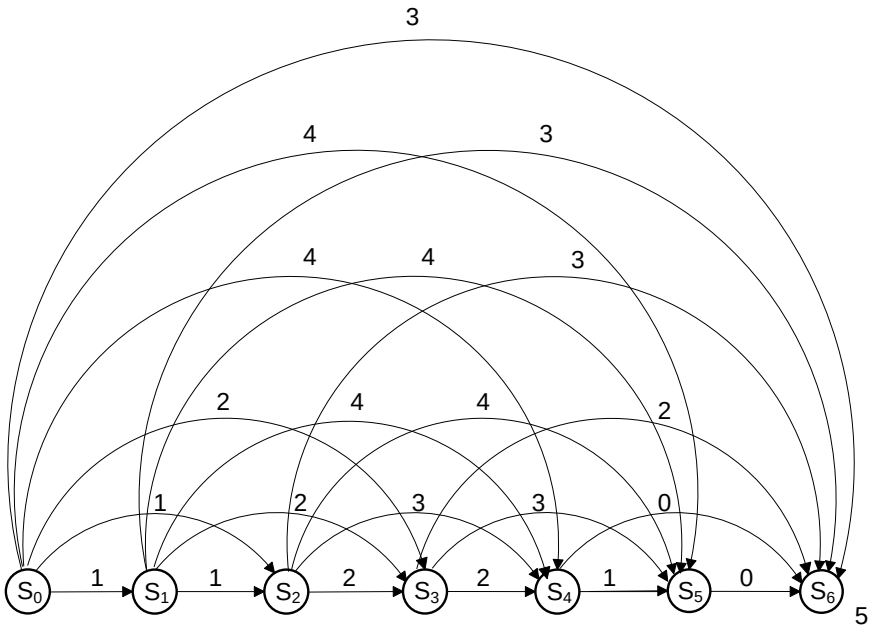


Figure 47: The graph representing sensors arrangement in Figure 46

Using dynamic programming, by calculating $L'(j,k)$ for all values of j,k, an optimal feasible solution can be calculated with time complexity $O(n^3)$. By storing the vertex S_i for which $L'(i,k-1)+w(S_i,S_j)$ gets its minimum value for the calculation of $L'(j,k)$, the optimal path P can be calculated in $O(n^3)$ time as well.

The vertices on the path would then represent the sensors that remain in their initial position. We can assign final positions to the remaining sensors in $O(n)$ time by scanning the barrier from the beginning and filling the gaps with the remaining sensors, which completes the proof. ■

6.2 The MinNum Problem on Multiple Barriers

In this section, we first consider the problem of covering two barriers with a set of n homogeneous sensor nodes with sensing ranges r. The barriers are two disjoint line segments, B_1 and B_2, on the same infinite line with lengths L_1 and L_2 respectively.

We develop centralized algorithms to minimize the number of sensors moved so as to provide maximum coverage in all the possible scenarios: $L_1+L_2 > 2rn$, $L_1+L_2 = 2rn$ and $L_1 + L_2 < 2rn$. We then extend our results for the coverage of any number of barriers.

6.2.1 $L_1 + L_2 = 2rn$

Theorem 6.2.1. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Also, let B_1 and B_2 be two disjoint line segments with lengths L_1 and L_2 respectively; i.e. $B_1 = [0, L_1]$ and $B_2 = [L_1 + g, L_1 + L_2 + g]$, where $g > 0$ is the gap length between the two barriers B_1 and B_2, and $L_1+L_2 = 2rn$. There is an $O(n^3)$ algorithm that solves the MinNum optimization problem of covering two line segments B_1 and B_2, so that the number of sensors moved is minimized.*

Proof. We consider two possible scenarios:

1) Both L_1 and L_2 are multiples of $2rn$. i.e $\exists i \in \mathbb{N}, \exists j \in \mathbb{N}, L_1 = 2ri \wedge L_2 = 2rj$. In this case complete coverage is possible, and all sensors have predetermined positions. Thus, sensors with initial positions in $\{r, 3r, \ldots, (2i-1)r\} \cup \{L_1 + g + r, L_1 + g + 3r, \ldots, L_1 + g + (2j-1)r\}$ remain stationary, and the remaining sensors should be moved to fill in the gaps in between. Clearly, this can be done in linear time.

2) Neither L_1 nor L_2 is multiple of $2rn$; i.e. $\nexists i \in \mathbb{N}, \nexists j \in \mathbb{N}, L_1 = 2ri \wedge L_2 = 2rj$. In this case, complete coverage is not possible, since there is at least one sensor that provides partial coverage, covering a part of the gap between the two barriers. Here we try to achieve maximum possible coverage while minimizing the number of sensors moved. We consider two scenarios depending on the length of the gap g. First, we consider the case where it is feasible for a sensor to cover both barriers at the same time ($g < 2r$). Then, we consider the case where any sensor can only cover one barrier ($g \geq 2r$).

i) $g < 2r$. To achieve maximum coverage, we consider the fraction of a sensor size that would cover the gap (is wasted), and therefore trying to minimize it. Let n_1 and n_2 be the number of sensors that fit on B_1 and B_2, respectively; i.e. $n_1 = \lfloor \frac{L_1}{2r} \rfloor$ and $n_2 = \lfloor \frac{L_2}{2r} \rfloor$. Also, let f_1 and f_2 be the fractional parts of barriers B_1 and B_2; i.e. $f_1 = L_1 - n_1 * 2r$ and $f_2 = L_2 - n_2 * 2r$.

Clearly, at least n_1 sensors should cover B_1 and at least n_2 sensors should cover B_2. The remaining one sensor, S, could cover B_1, B_2 or both B_1 and B_2 providing maximum coverage $L_1 + n_2 * 2r$, $n_1 * 2r + L_2$, or $n_1 * 2r + (2r - g) + n_2 * 2r$ respectively. First we assume $f_1 > f_2$. Therefore, S should either cover B_1 only, or should cover both B_1 and B_2.

If $f_1 > 2r - g$, maximum coverage is achieved by using $n_1 + 1$ sensors to cover B_1 completely, and using n_2 to cover B_2. This can be achieved in $O(n^3)$ using the algorithms in Theorems 6.1.11 and 6.1.10, respectively.

If $f_1 < 2r - g$, maximum coverage is achieved only if there exists one sensor S that completely covers the gap (covers both B_1 and B_2). We show that any optimal solution that minimizes the number of sensors moved while maximizing the coverage on virtual barrier $[0, L_1 + L_2 + g]$ has a sensor that completely covers $[L_1, L_1 + g]$. Assume there is an optimal solution with maximal coverage $2rn$ on $[0, L_1 + L_2 + g]$ with no sensor that completely covers the gap. Therefore, the number of sensors that are to the left of $L_1 + g$ is less than or equal to $\lfloor \frac{L_1+g}{2r} \rfloor = \lfloor \frac{2rn_1+f_1+g}{2r} \rfloor = n_1$. Furthermore, the number of sensors that are to the right of L_1 is less than or equal to $\lfloor \frac{L_2+g}{2r} \rfloor = \lfloor \frac{2rn_2+f_2+g}{2r} \rfloor = n_2$ (Note that we assumed $f_2 < f_1$). This means that there are maximum $n_1 + n_2 = n - 1$ sensors used to cover the barrier $[0, L_1 + L_2 + g]$ which contradicts the fact that the maximal coverage is equal to $2rn$. Therefore, any optimal solution that minimizes the maximum sensor movement while maximizing the coverage on

virtual barrier $[0, L_1 + L_2 + g]$ has a sensor that completely covers $[L_1, L_1 + g]$, and this can be obtained in $O(n^3)$ using the algorithm in Theorem 6.1.10.

If $f_1 = 2r - g$, compare two optimal solutions for both cases above, and select the one with the smaller number of sensors moved.

The case where $f_2 > f_1$ is similar to $f_1 < f_2$, except for that B_2 should be covered instead of B_1. When $f_1 = f_2$, the optimal solution is the better of the solutions for $f_1 < f_2$ and $f_1 > f_2$.

ii) $g \geq 2r$. Let $n_1 = \lfloor \frac{L_1}{2r} \rfloor$ and $f_1 = L_1 - 2rn_1$. Consider the following subcases:

(a) $f_1 > f_2$. Use the $O(n^3)$ algorithm in Theorem 6.1.11 for the MinNum optimization problem with $n_1 + 1$ sensors on B_1, and the $O(n^3)$ algorithm in Theorem 6.1.10 for the MinNum optimization problem with $n - n_1 - 1$ sensors on B_2.

(b) $f_1 < f_2$. Use the $O(n^3)$ algorithm in Theorem 6.1.10 for the MinNum optimization problem with n_1 sensors on B_1, and the $O(n^3)$ algorithm in Theorem 6.1.11 for the MinNum optimization problem with $n - n_1$ sensors on B_2.

(c) $f_1 = f_2$. Execute both algorithms for cases where $f_1 > f_2$ and $f_1 < f_2$ with time complexity $O(n^3)$. Compare the optimal solutions of both algorithms. The solution with the minimum number of sensors moved is the optimal solution.

■

Corollary 6.2.2. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(mn^3)$ algorithm that solves the MinNum optimization problem of covering m line segments B_1 and B_2, ..., B_m with $\sum_{i=1}^{m} |B_i| = 2rn$ so that the number of sensors moved is minimized.*

Proof. If the barrier lengths are multiples of $2r$, the sensors' positions are predetermined and the final positions can be assigned in $O(n)$ time. Otherwise, a barrier with the maximum fractional part should be overcovered and the rest are undercovered. There are at most m barriers with maximum fractional part and thus this can be done in $O(mn^3)$. Furthermore, if it is possible that a sensor covers more than one barrier using the similar argument used in the proof of the theorem, an optimal solution for the coverage of the virtual barrier $[0, \sum_{i=1}^{n-1} |B_i| + |g_i| + |B_n|]$ is the optimal solution. ■

6.2.2 $L_1 + L_2 > 2rn$

It is obvious that when the sum of the barriers' lengths is greater than the sensing range provided by all the sensors, complete coverage is not possible. Therefore, we consider the problem of maximizing the possible coverage, while minimizing the number of sensors moved, and we study the contiguous as well as the non-contiguous MinNum optimization problem.

For the contiguous MinNum optimization problem, we distinguish two cases depending on whether maximal coverage of $2rn$ is feasible or not. Let n_1 and n_2 be the maximum number of sensors than can be completely contained in B_1 and B_2 respectively. If $n > n1 + n2$, the maximal coverage is not possible. However this can only happen if none of L_1 and L_2 are multiples of $2r$, and $n = n_1 + n_2 + 1$. This case can be handled similar to the case where $L_1 + L_2 = 2rn$, where one of the barriers would be undercovered and the other one would be overcovered. Thus, we only consider the scenario where maximal coverage of $2rn$ is feasible, $n_1 + n_2 \geq n$. In other words we have $L_1 + L_2 \geq 2r(n + 1)$.

Theorem 6.2.3. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. Also, let B_1 and B_2 be two disjoint line segments with*

lengths L_1 and L_2 respectively; i.e. $B_1 = [0, L_1]$ and $B_2 = [L_1 + g, L_1 + L_2 + g]$, where $g > 0$ is the gap length between the two barriers B_1 and B_2, and $L_1 + L_2 \geq 2r(n+1)$. There is an $O(n^2)$ algorithm that solves the MinNum optimization problem of covering two line segments B_1 and B_2, so that the number of sensors moved is minimized.

Proof. Since $R < L_1 + L_2$, and $L_1 + L_2 \geq 2r(n+1)$, maximal coverage of $2rn$ is feasible, and thus only sensors completely contained in either L_1 or L_2 can remain fixed. Let T_1 and T_2 be the set of sensors that are completely contained in B_1 and B_2 respectively; i.e. $T_1 = \{S_i | S_i \in S \wedge r \leq x_i \leq L_1 - r\}$ and $T_2 = \{S_i | S_i \in S \wedge L_1 + g + r \leq x_i \leq L_1 + L_2 + g - r\}$. The optimal solution to the contiguous MinNum optimization problem can be obtained by using the algorithm in Theorem 6.1.7 to sloe the contiguous MinNum optimization problem for the set of sensors T_1 to cover B_1 and the set of sensors T_2 to cover B_2 with time complexity $O(|T_1|^2 + |T_2|^2) = O(n^2)$. ■

Corollary 6.2.4. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(n^2)$ algorithm that solves the contiguous MinNum optimization problem of covering m line segments B_1 and B_2, ..., B_m with $\sum_{i=1}^{m} |B_i| \geq 2r(n+m)$ so that the number of sensors moved is minimized.*

We now study the non-contiguous MinNum optimization problem.

Theorem 6.2.5. *Let S be the set of sensors $S_1, S_2, ..., S_n$ with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$ with $L_1 + L_2 > 2rn$. There is an $O(n^3)$ algorithm that solves the non-contiguous MinNum optimization problem of covering two line segments $B_1 = [0, L_1]$ and $B_2 = [L_1 + g, L_1 + L_2 + g]$ so that the number of sensors moved is minimized.*

Proof. Since $L_1 + L_2 > 2rn$, complete coverage is not possible. Therefore, the goal is to provide maximum coverage $2rn$ while minimizing the number of sensors moved. However, if L_1 and L_2 are not multiples of $2rn$ and $L_1 + L_2$ is close to $2rn$, $2rn <$

$L_1 + L_2 < 2r(n+1)$, coverage of $2rn$ is not feasible, since one of the sensors would partially cover the gap in between B_1 and B_2. In this case, use the $O(n^3)$ algorithm in Theorem 6.2.1. Otherwise, coverage of $2rn$ is possible and the problem can be reduced to the problem of finding the maximum-hop path P with total weight at least n on G, where the vertices in G are the sensors in $S' = \{S_i \in S | r \leq x_i \leq L_1 - r \vee L_1 + g + r \leq x_i \leq L_1 + L_2 + g - r\}$ and the weight of a directed edge $E_{i,j} = (S_i, S_j)$, $j > i$, indicates the number of non-intersecting sensors that can fit between S_i and S_j including S_i, if both S_i and S_j are stationary. More formally, the weight of and edge can be described as follows:

$$w(S_i, S_j) = \begin{cases} w'(S_i, S_j) + 1 & \text{if } j \leq n, \\ w'(S_i, S_j) & \text{if } j = n+1. \end{cases}$$

$$w'(S_i, S_j) = \begin{cases} \lfloor \frac{le(S_j) - re(S_i)}{2r} \rfloor & \text{if } x_j \leq L_1 - r \vee x_i \geq L_1 + g + r, \\ \lfloor \frac{L_1 - re(S_i)}{2r} \rfloor + \lfloor \frac{le(S_j) - L_1 - g}{2r} \rfloor & \text{if } x_i \leq L_1 - r \wedge x_j \geq L_1 + g + r. \end{cases}$$

Clearly, G can be constructed in quadratic time, and once constructed, we can use the dynamic programming approach used in Theorem 6.1.10, to get the optimal solution with time complexity $O(n^3)$. ■

Corollary 6.2.6. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(n^3)$ algorithm that solves the non-contiguous MinNum optimization problem of covering m line segments B_1 and B_2, ..., B_m with $\sum_{i=1}^{m} |B_i| \geq 2r(n+m)$ so that the number of sensors moved is minimized.*

Proof. The set of vertices in G can be found in linear time by checking the sensors coordinate and verifying if they fall completely on one of the barriers. Once the

vertices are determined, weights can be determined in constant time, and thus G can be constructed in quadratic time. The problem of finding the maximum-hop path P with total weight at least n on G is independent of the number of barriers m. ■

6.2.3 $L_1 + L_2 < 2rn$

In this scenario, since the sum of the barrier lengths is smaller than the maximal coverage provided by all sensors, not all sensors need to participate in the coverage problem. Therefore, we need to move some of the sensors in order to cover the two barriers while minimizing the number of sensors moved.

Theorem 6.2.7. *Let $S = \{S_1, S_2, ..., S_n\}$ be the set of sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$ with $L_1 + L_2 < 2rn$. There is an $O(n^3)$ algorithm that solves the MinNum optimization problem of covering two line segments $B_1 = [0, L_1]$ and $B_2 = [L_1 + g, L_1 + L_2 + g]$ so that the number of sensors moved is minimized.*

Proof. Let n_1 and n_2 be the minimum number of sensors that are needed to completely cover barriers B_1 and B_2, respectively; i.e. $n_1 = \lceil \frac{L_1}{2r} \rceil$ and $n_2 = \lceil \frac{L_2}{2r} \rceil$. We consider the following scenarios:

i) $n_1 + n_2 = n + 1$. Thus, we have $n_1 + n_2 = n - 1$, and complete coverage is not possible. Therefore, we aim at maximizing the coverage while minimizing the number of sensors moved. This can be done in $O(n^3)$ using the algorithm in Theorem 6.2.1.

ii) $n_1 + n_2 \leq n$. In this case complete coverage is possible, and the problem reduces to the problem of maximum-hop path P with total weight at most n on G, where the vertices in G are the sensors in $S' = \{S_i \in S | -r < x_i < L_1 + r \vee L_1 + g - r < x_i < L_1 + L_2 + g + r\}$ and the weight of a directed edge $E_{i,j} = (S_i, S_j)$, $j > i$,

indicates the minimum number of sensors needed to cover the gap between S_i and S_j including S_i, if both S_i and S_j are stationary. More formally, the weight of and edge can be described as follows:

$$w(S_i, S_j) = \begin{cases} w'(S_i, S_j) + 1 & \text{if } j \leq n, \\ w'(S_i, S_j) & \text{if } j = n+1. \end{cases}$$

$$w'(S_i, S_j) = \begin{cases} \lceil \frac{le(S_j) - re(S_i)}{2r} \rceil & \text{if } x_j < L_1 + r \vee x_i > L_1 + g - r, \\ 0 & \text{if } L_1 - r \leq x_i < L_1 + g - r \\ & \quad < x_j \leq L_1 + g + r, \\ \lceil \frac{L_1 - re(S_i)}{2r} \rceil & \text{if } x_i < L_1 - r \leq x_j \leq L_1 + g + r, \\ \lceil \frac{le(S_j) - L_1 - g}{2r} \rceil & \text{if } L_1 - r \leq x_i \\ & \quad \leq L_1 + g + r < x_j, \\ \lceil \frac{L_1 - re(S_i)}{2r} \rceil + \lceil \frac{le(S_j) - L_1 - g}{2r} \rceil & \text{if } x_i < L_1 - r \wedge x_j > L_1 + g + r. \end{cases}$$

The graph G can be constructed with time complexity, and then the $O(n^3)$ algorithm in Theorem 6.1.11 can be used to obtain the optimal solution.

■

Corollary 6.2.8. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $x_1 \leq x_2 \leq ... \leq x_n$. There is an $O(n^3)$ algorithm that solves the Min-Num optimization problem of covering m line segments B_1 and B_2, ..., B_m with $\sum_{i=1}^{m} |B_i| \leq 2r(n-m)$ so that the number of sensors moved is minimized.*

6.3 The MinNum Problem on a Circle

In this section, we assume that the sensors are arbitrarily dispersed on the circumference of a circular barrier C, with diameter d, centered at $o = (0, 0)$. The goal is to move the fewest number of sensors possible so as to achieve maximal coverage of circle C. First, we consider the case where sensors have arbitrary ranges, and we show it to be NP-complete. Then we present polynomial algorithms for sensors with identical ranges.

6.3.1 Unequal Range Sensors

We show that when sensor nodes have unequal ranges, all variations of the MinNum optimization problem on a circle barrier is NP-Complete.

Theorem 6.3.1. *Let $S_1, S_2, \ldots, S_n$ be sensors with arbitrary ranges $r_1, r_2, \ldots, r_n$ in initial positions $0 \leq \theta_1 \leq \theta_2 \leq ... \leq \theta_n \leq 2\pi$. The MinNum optimization problem on a circle barrier $C = (o, \frac{d}{2})$, where $L = \sum_{i=1}^{n} 2r_i$ is NP-hard.*

Proof. We prove it by reducing the *Partition Problem* [GJ90] into the MinNum optimization problem. Let $a_1 \geq a_2 \geq ... \geq a_n$ be integers, and let $a = \sum_{i=1}^{n} a_i$. Given an instance of the partition problem, we transform it into the MinNum optimization problem on a circular barrier $C = (o, \frac{d}{2})$ with the sensor set $S = \{S_1, S_2, ..., S_{n+1}, S_{n+2}\}$, where $L = \pi d = a + 4\delta$. For every $1 \leq i \leq n$, S_i with sensing range $\frac{a_i}{2}$ is initially located at angular position $\frac{\pi}{2}$. Also, there are two sensors S_{n+1} and S_{n+2}, with sensing ranges $\delta \leq \frac{1}{2}$, with initial positions $\frac{\pi}{2}$ and $\frac{3\pi}{2}$ respectively (see Figure 48). If there is a set of indices J, such that $\sum_{i \in J} a_i = \frac{a}{2}$, there is a solution to the MinNum optimization problem such that the sensors with indices in J cover the clockwise segment between S_{n+1} and S_{n+2} and the rest cover the counterclockwise segment between S_{n+1} and S_{n+2}, and the maximum number of sensors moved is n.

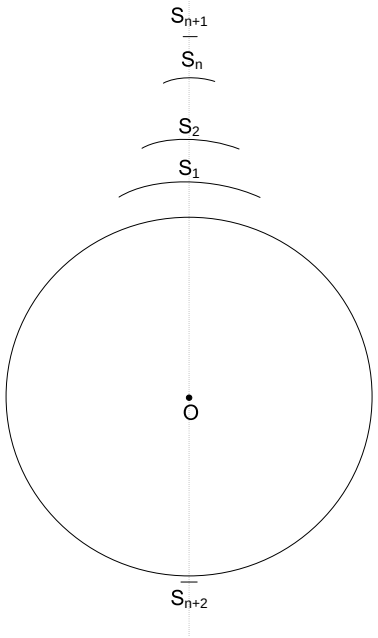


Figure 48: Arrangement of sensors for proving the NP-completeness of the MinNum optimization problem for unequal sensor ranges on a circle barrier $C = (o, \frac{d}{2})$.

If such a partition does not exist, $n + 1$ sensors have to move.

Thus, if there is an algorithm that can solve the MinNum optimization problem on a circle $C = (o, \frac{d}{2})$, where $R = L$, we can determine whether the partition problem has a solution. Clearly, the transformation from the partition problem to the MinNum optimization problem is polynomial. ■

It is easy to show that the problem for $R < L$ and $R > L$ remains NP-complete, by using the same proof above and considering a circle of circumference $R + \epsilon$ and $R - \epsilon$ respectively, where ϵ is less than twice the minimum range of sensors. Observe that the argument holds for the non-contiguous coverage as well as the contiguous coverage for the case $R < L$.

6.3.2 Equal Range Sensors

In view of the NP-completeness, we consider sensors $\{S_1, S_2, ..., S_n\}$ with identical sensing range r. Every sensor S_i is located at polar coordinate $p_i = (\theta_i, \frac{d}{2})$. Furthermore, assume that $0 \leq \theta_1 \leq \theta_2 \leq \cdots \leq \theta_n \leq 2\pi$. Since all the sensors are located on the circumference of the circle, throughout this study, we use the angle θ in their polar coordinate as the nodes' positions.

Let $cw\_dist(\alpha, \beta)$ be the angular distance between α and β, when we traverse the circle from $(\alpha, \frac{d}{2})$ to $(\beta, \frac{d}{2})$ clockwise (see Figure 49).

$$cw\_dist(\alpha, \beta) = \begin{cases} \alpha - \beta & \text{if } \alpha \geq \beta, \\ 2\pi - (\beta - \alpha) & \text{if } \alpha < \beta. \end{cases}$$

Analogously, let $ccw\_dist(\alpha, \beta)$ be the counterclockwise angular distance between α and β; i.e. $ccw\_dist(\alpha, \beta) = 2\pi - cw\_dist(\alpha, \beta)$.

We study all possible scenarios: $2rn = L$, $2rn < L$ and $2rn > L$, where $L = \pi d$. The only case where complete coverage is not possible is when $2rn < L$. Therefore, when $2rn < L$, we consider both contiguous and non-contiguous coverage.

Lemma 6.3.2. *Let $S_1, S_2, ..., S_n$ be sensors with sensing ranges $r_1, r_2, ..., r_n$ in initial positions $0 \leq \theta_1 \leq \theta_2 \leq ... \leq \theta_n \leq 2\pi$. The MinNum optimization problem has at least one stationary sensor in all cases.*

Proof. Assume that there is an optimal solution in which all sensors have been relocated. Rotate the sensors clockwise until one of the sensors is at its initial position. This assignment of final positions provides exactly the same coverage as the optimal solution with one more stationary sensor, which contradicts the assumption that the solution was optimal. ■

Using Lemma 6.3.2, we show that the optimal solution to the barrier coverage

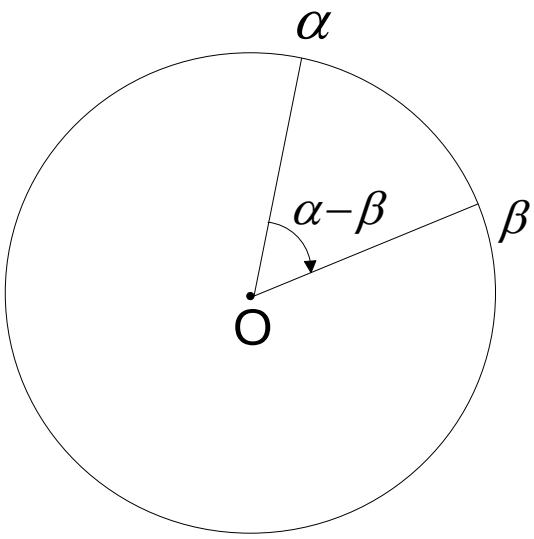


Figure 49: Clockwise distance between $(\alpha, \frac{d}{2})$ and $(\beta, \frac{d}{2})$.

problem on a circle can be obtained by converting it to the barrier coverage problem on a line segment by breaking the circle at one of the sensor nodes.

Lemma 6.3.3. *Let $S_1, S_2, ..., S_n$ be sensors with identical range r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$. Let P_i be the MinNum optimization problem on the line segment $[0, \pi d]$ with sensors $\{S'_1, S'_2, ..., S'_n\}$ with identical range r at initial positions $r \leq r + \frac{d}{2} ccw\_dist(\theta_{(i+1) \mod n}, \theta_i) \leq \cdots \leq r + \frac{d}{2} ccw\_dist(\theta_{(i+n-1) \mod n}, \theta_i)$. The MinNum optimization problem on a circle $C = (o, \frac{d}{2})$ has a solution in a set of solutions to the MinNum optimization problems P_i.*

Proof. Let P be an optimal solution on the circle C. Furthermore, let $T \subseteq S$ be the set of stationary sensors in P. Note that according to Lemma 6.3.2, $T \neq \emptyset$. For every S_i in T, since S_i is stationary, converting the problem into P_i on line segment $[0, L]$ would preserve the gaps exactly the way they are on the circle, and therefore P is also an optimal solution for the problem P_i. Thus, by solving all the optimization problems P_i, we can get an optimal solution on the circle. ■

Corollary 6.3.4. *Let $S_1, S_2, ..., S_n$ be sensors with identical range r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_n \leq 2\pi$. There is a polynomial algorithm for the MinNum optimization problem on a circular barrier $C = (o, \frac{d}{2})$ for all possible scenarios: $R < L$, $R = L$, $R > L$ with time complexities $O(n^4)$, $O(n^2)$, and $O(n^4)$ respectively.*

We showed that the optimal solution for the MinNum optimization problem P on a circular barrier is among the optimal solutions for the MinNum optimization problems P_i on line barriers, where the circle is broken into line at sensor S_i. We proceed to show better algorithms for the contiguous MinNum optimization problem when $R < L$, as well as the and non-contiguous MinNum optimization problem when $2R \geq L$.

Theorem 6.3.5. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq ... \leq \theta_n \leq 2\pi$ with $\pi d > 2rn$. There is an $O(n^2)$ algorithm that solves the contiguous MinNum optimization problem on a circle $C = (o, \frac{d}{2})$.*

Proof. Let ω be the angular coverage of sensors; i.e. $\omega = \frac{4r}{d}$. Using Lemma 6.3.2, at least one of the sensors can remain stationary. For every sensor S_i, find the sets $T_{cw,i}$ and $T_{ccw,i}$ of sensors that are at angular distances multiples of ω from S_i in both clockwise and counterclockwise order; i.e. $T_{cw,i} = \{S_j \in S | \exists k \in \mathbb{N}, 0 \leq k \leq n-1, cw\_dist(\theta_i, \theta_j) = k\omega\}$ and $T_{ccw,i} = \{S_j \in S | \exists k \in \mathbb{N}, 0 \leq k \leq n-1, ccw\_dist(\theta_i, \theta_j) = k\omega\}$. The maximum number of sensors that can stay fixed once S_i is fixed is $\max(|T_{ccw,i}|, |T_{cw,i}|)$ which can be calculated in linear time and the optimal solution is the one that maximizes the number of fixed sensors among all S_is and can be calculated in quadratic time. Once the fixed sensors are determined, the non-covered intervals can be covered with the remaining sensors in linear time which results in total time complexity $O(n^2)$. ■

The maximal non-contiguous coverage problem can be divided into two subproblems: (1) Finding a set of non-intersecting sensors as the fixed sensors (2) Assigning

final positions to the non-fixed sensors. In order to solve the non-contiguous MinNum optimization problem, one should find a maximal set of fixed sensors, such that the remaining sensors fit in the gaps. Clearly, any optimal solution has at most as many fixed sensors as the size of a maximal set of non-intersecting sensors. Therefore, if the barrier is big enough the problem reduces to the problem of finding a maximal set of non-intersecting sensors on a circle. Using the same argument which was used in Lemma 6.1.8, we can show that if $L = \pi d \geq 4rn$, once the maximal set of non-intersecting sensors is found, the remaining sensors can fit in between the non-covered intervals so as to provide maximal coverage of $2rn$. As a result, we distinguish two cases: $2rn < \pi d < 4rn$ and $\pi d \geq 4rn$, and we give a better algorithm than the one given in Corollary 6.3.4, for the case where $\pi d \geq 4rn$.

Theorem 6.3.6. *Let $S_1, S_2, ..., S_n$ be sensors with identical ranges r in initial positions $0 \leq \theta_1 \leq \theta_2 \leq ... \leq \theta_n \leq 2\pi$ with $\pi d \geq 4rn$. There is an optimal $O(n)$ algorithm that solves the non-contiguous MinNum optimization problem on a circle $C = (o, \frac{d}{2})$.*

Proof. Partition the set of sensors S into subsets N_i such that for every sensor $S_j \in N_i$ and $S_{(j+1) \mod n} \in N_i$, $S_j \cap S_{(j+1) \mod n} \neq \emptyset$, and for every N_i and N_j, $\forall S_k \in N_i, S_l \in N_j, S_k \cap S_l = \emptyset$. A maximal independent set for every N_i can be calculated using the greedy algorithm for the activity selection in time $O(|N_i|)$. A set of non-intersecting sensors is thus the union of maximal independent sets of N_is. Clearly, this approach has linear time complexity. ■

6.4 Conclusion

We studied the MinNum optimization problem, as well as the contiguous MinNum optimization problem, and the non-contiguous MinNum optimization problem. When sensors had unequal ranges, we showed that all three optimization problems on a line

segment barrier as well as circular barriers were NP-hard. In contrast, when sensors had equal ranges, we presented several efficient algorithms to solve the optimization problems stated above. All our algorithms were centralized: they were given initial positions of sensors and they calculated optimized final positions. A summary of the complexities of the algorithms given for all cases for sensors with identical ranges r, is given in Table 4. Furthermore, since $O(n)$ is a trivial lower bound for the time complexities of all the algorithms presented in this chapter, all our linear algorithms are optimal.

Table 4: Algorithm complexities for the MinNum problem for homogeneous sensors.

	Contiguous	Non-contiguous
Infinite line	$O(n^2)$	$O(n)$
Line segment $R = 2rn = L$	$O(n)$	N.A.
Line segment $R = 2rn < L$	$O(n^2)$	$O(n^3)$
Line segment $R = 2rn > L$	$O(n^3)$	N.A.
m line segments $R = 2rn = \sum_{i=1}^{m} L_i$	$O(mn^3)$	N.A.
m line segments $2r(n+m) \leq \sum_{i=1}^{m} L_i$	$O(n^3)$	$O(n^2)$
m line segments $2r(n-m) \geq \sum_{i=1}^{m} L_i$	$O(n^3)$	N.A.
Circular barrier $c_r n = L = 2\pi r$	$O(n^2)$	N.A.
Circular barrier $cr_n < L = 2\pi r < 2c_r n$	$O(n^2)$	$O(n^4)$
Circular barrier $2c_r n \leq L = 2\pi r$	$O(n^2)$	$O(n)$
Circular barrier $c_r n > L = 2\pi r$	$O(n^4)$	N.A.

Chapter 7

Conclusions and Future Work

In this thesis, we studied the fundamental problem of routing, backbone formation and the barrier coverage problem in wireless ad hoc and sensor networks. We introduced a new local learning routing algorithm which learns about the existence of an obstacle or void using negative feedback from its neighbors. Using this information, the algorithm diverts from the greedy path when it gets close to the obstacle region. Through simulations we showed that after five retrials our algorithm has a delivery ratio of almost 100% and an average path length which is very close to that of the greedy algorithm. The weighted distance function that we used in the algorithm is a heuristic and it can be further improved by taking some other information into consideration. Furthermore, analyzing the effect of the river routing algorithm on load balancing and congestion control is an interesting problem, since unlike the GFG routing algorithm which usually hugs the border of void especially in the case of large voids, river routing selects different routes each time.

Then, we considered the problem of forming efficient local data gathering and dissemination backbones. For the class of UDGs and QUDGs, we presented a local CDS approximation algorithm with constant approximation ratio. We also considered the

more realistic model of DGs and we presented a local SCDAS approximation algorithm with constant approximation ratio. Through extensive simulations, we showed that our algorithms outperform the best existing algorithms in the literature in terms of the CDS size. Although our algorithms construct very thin CDSs and SCDASs in practice, the tiling that we use does not lead to a small theoretical approximation ratio. Finding a simple local algorithm with a small approximation ratio and that also performs very well in practice remains an interesting open problem.

Finally, we considered the problem of barrier coverage using mobile sensors. We presented centralized algorithms to instruct sensors to move to final positions so as to achieve maximum coverage of the barrier. We considered two different aspects of minimizing energy while providing maximum coverage, MinMax and MinNum. We studied the barrier coverage problem when barriers are multiple line segment as well as circular barriers. We considered all possible scenarios depending on whether or not the sensors have equal sensing ranges, whether or not complete coverage is feasible, and if complete coverage is not feasible, the maximal coverage is contiguous or non-contiguous. For all scenarios we either give efficient polynomial algorithms or we show the problem to be NP-hard. Neither MinNum nor MinMax has been studied in the case where there are several sets of sensor ranges, and these problems still remain open. Furthermore, one can consider MinNum and MinMax on different barrier types such as polygons. Finally, for the MinMax problem, the case where sensors are not initially on the perimeter of the circular barrier needs to be investigated.

Bibliography

[ACR07] T. Acharya, S. Chattopadhyay, and R. Roy. Energy-aware virtual backbone tree for efficient routing in wireless sensor networks. In *Proceedings of ICNS'07*, pages 96–101, 2007.

[ASSC02] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[AWF02] K. M. Alzoubi, P. J. Wan, and O. Frieder. Message-optimal connected dominating sets in mobile ad hoc networks. In *Proceedings of MobiHoc'02*, pages 157–164, 2002.

[AY05] K. Akkaya and M. F. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349, 2005.

[Bas99] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of ISPAN'99*, pages 310–315, 1999.

[BBH+08] B. K. Bhattacharya, B. Burmester, Y. Hu, E. Kranakis, Q. Shi, and A. Wiese. Optimal movement of mobile sensors for barrier coverage of a planar region. In *Proceedings of COCOA'08*, volume 5165 of *LNCS*, pages 103–115, 2008.

[BBSK07] P. Balister, B. Bollobas, A. Sarkar, and S. Kumar. Reliable density estimates for coverage and connectivity in thin strips of finite length. In *Proceedings of MobiCom'07*, pages 75–86, 2007.

[BGJ05] J. Bruck, J. Gao, and A. Jiang. Map: medial axis based geometric routing in sensor networks. In *Proceedings of MobiCom'05*, pages 88–102, 2005.

[BMP04] S. Basagni, M. Mastrogiovanni, and C. Petrioli. A performance comparison of protocols for clustering and backbone formation in large scale ad hoc network. In *Proceedings of MASS'04*, pages 70–79, 2004.

[BMSU99] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proceedings of DIAL-M'99*, pages 48–55, 1999.

[CCJ90] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.

[CDF+08] J. Czyzowicz, S. Dobrev, T. Fevens, H. González-Aguilar, E. Kranakis, J. Opatrny, and J. Urrutia. Local algorithms for dominating and connected dominating sets of unit disk graphs with location aware nodes. In *Proceedings of LATIN'08*, volume 4957 of *LNCS*, pages 158–169, 2008.

[CDK+06] E. Chávez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, and J. Urrutia. Local construction of planar spanners in unit disk graphs with irregular transmission ranges. In *Proceedings of LATIN'06*, volume 3887 of *LNCS*, pages 286–297, 2006.

[Chv79] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[CKK+09] J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, J. Opatrny, L. Stacho, J. Urrutia, and M. Yazdani. On minimizing the maximum sensor movement for barrier coverage of a line segment. In *Proceedings of ADHOC-NOW'09*, volume 5793 of *LNCS*, pages 194–212, 2009.

[CKK+10] J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, J. Opatrny, L. Stacho, J. Urrutia, and M. Yazdani. On minimizing the sum of sensor movements for barrier coverage of a line segment. In *Proceedings of ADHOC-NOW'10*, volume 6288 of *LNCS*, pages 29–42, 2010.

[CKL07] A. Chen, S. Kumar, and T. H. Lai. Designing localized algorithms for barrier coverage. In *Proceedings of MobiCom'07*, pages 63–74, 2007.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms 2nd edition*. MIT Press, 2001.

[DHM+09] E. D. Demaine, M. Hajiaghayi, H. Mahini, A. S. Sayedi-Roshkhar, S. Oveisgharan, and M. Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5(3):1–30, 2009.

[DKN08] S. Durocher, D. G. Kirkpatrick, and L. Narayanan. On routing with guaranteed delivery in three-dimensional ad hoc wireless networks. In *Proceedings of ICDCN'08*, volume 4904 of *LNCS*, pages 546–557, 2008.

[DSW02] S. Datta, I. Stojmenovic, and J. Wu. Internal node and shortcut based routing with guaranteed delivery in wireless networks. *Cluster Computing*, 5(2):169–178, 2002.

[DW04] F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Transactions on Parallel Distributed Systems*, 15(10):908–920, 2004.

[Fei98] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[FGG+05] Q. Fang, J. Gao, L. J. Guibas, V. Silva, and L. Zhang. Glider: Gradient landmark-based distributed routing for sensor networks. In *Proceedings of INFOCOM'05*, pages 339–350, 2005.

[Fin87] G. G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. Technical report, University of Southern California, ISI, 1987.

[Fis85] P. C. Fishburn. *Interval Orders and Interval Graphs*. J. Wiley, New York, 1985.

[GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[GSB03] S. Giordano, I. Stojmenovic, and L. Blazevic. Position based routing algorithms for ad hoc networks: a taxonomy. In *Proceedings of AHWN'03*, pages 103–136, 2003.

[HJB04] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *Proceedings of SenSys'04*, pages 134–147, 2004.

[HT03] C. F. Huang and Y. C. Tseng. The coverage problem in a wireless sensor network. In *Proceedings of WSNA'03*, pages 115–121, 2003.

[JRS02] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193–205, 2002.

[KGKS05] Y. J. Kim, R. Govindan, B. Karp, and S. Shenker. On the pitfalls of geographic face routing. In *Proceedings of DIALM-POMC'05*, pages 34–43, 2005.

[KK00] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of MobiCom'00*, pages 243–254, 2000.

[KLA05] S. Kumar, T. H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proceedings of MobiCom'05*, pages 284–298, 2005.

[KLB04] S. Kumar, T. H. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In *Proceedings of MobiCom'04*, pages 144–158, 2004.

[KMNO09] H. Kassaei, M. Mehrandish, L. Narayanan, and J. Opatrny. A new local algorithm for backbone formation in ad hoc networks. In *Proceedings of PE-WASUN'09*, pages 49–57, 2009.

[KMNO10] H. Kassaei, M. Mehrandish, L. Narayanan, and J. Opatrny. Efficient algorithms for connected dominating sets in ad hoc networks. In *Proceedings of WCNC'10*, pages 1–6, 2010.

[KN10] H. Kassaei and L. Narayanan. A new algorithm for backbone formation in ad hoc wireless networks of nodes with different transmission ranges. In *Proceedings of WiMob'10*, pages 83 – 90, 2010.

[KSU99] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proceedings of CCCG'99*, pages 51–54, 1999.

[KW05] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

[KWZ03] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proceedings of MobiHoc'03*, pages 267–278, 2003.

[KWZZ03] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proceedings of PODC'03*, pages 63–72, 2003.

[LS98] X. Lin and I. Stojmenovic. Geographic distance routing in ad hoc wireless networks. Technical report, 1998.

[LW07] C. Liu and J. Wu. Destination-region-based local minimum aware geometric routing. *IEEE International Conference on Mobile Ad hoc and Sensor Systems*, 1:1–9, 2007.

[LWS04] X. Y. Li, Y. Wang, and W. Z. Song. Applications of k-local MST for topology control and broadcasting in wireless ad hoc networks. In *Proceedings of INFOCOM'04*, pages 7–11, 2004.

[MBHI+95] M.V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.

[MH01] M. Mauve and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, 15(6):30–39, 2001.

[MKPS01] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M.B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *Proceedings of INFOCOM'01*, volume 3, pages 1380–1387, 2001.

[MNO11] M. Mehrandish, L. Narayanan, and J. Opatrny. Minimizing the number of sensors moved on line barriers. In *Proceedings of WCNC'11*, pages 1464–1469, 2011.

[NH05] T. Nieberg and J. L. Hurink. A PTAS for the minimum dominating set problem in unit disk graphs. In *Proceedings of WAOA'05*, volume 3879 of *LNCS*, pages 296–306, 2005.

[PDDB05] V. Paruchuri, A. Durresi, M. Durresi, and L. Barolli. Routing through backbone structures in sensor networks. In *Proceedings of ICPADS'05*, pages 397–401, 2005.

[PWW+07] M. A. Park, J. Willson, C. Wang, M. Thai, W. Wu, and A. Farago. A dominating and absorbent set in a wireless ad-hoc network with different transmission ranges. In *Proceedings of MobiHoc'07*, pages 22–31, 2007.

[SLX+10] A. Saipulla, B. Liu, G. Xing, X. Fu, and J. Wang. Barrier coverage with sensors of limited mobility. In *Proceedings of MobiHoc'10*, pages 201–210, 2010.

[SSB99] R. Sivakumar, P. Sinha, and V. Bharghavan. Cedar: a core-extraction distributed ad hoc routing algorithm. *IEEE Journal on Selected Areas in Communications*, 17:1454–1465, 1999.

[WAF04] P. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications - Discrete algorithms and methods for mobile computing and communications*, 9(2):141–149, 2004.

[WCLP06] G. Wang, G. Cao, and T. F. La Porta. Movement-assisted sensor deployment. *IEEE Transactions on Mobile Computing*, 5(6):640–652, 2006.

[WK08] A. Wiese and E. Kranakis. Local PTAS for dominating and connected dominating set in location aware unit disk graphs. In *Proceedings of WAOA'08*, volume 5426 of *LNCS*, pages 227–240, 2008.

[WL99] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of DIALM'99*, pages 7–14, 1999.

[Wu02] J. Wu. Extended dominating-set-based routing in ad hoc wireless networks with unidirectional links. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):866–881, 2002.

[YEG01] Y. Yu, D. Estrin, and R. Govindan. Geographical and energy-aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical report, University of California, Los Angeles: Computer Science Department, 2001.