Router Placement in Wireless Sensor Networks

M I C H A E L   A H L B E R G

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LECS-2005-94

Router Placement in Wireless Sensor Networks

Michael Ahlberg

Master of Science Thesis

Performed at
Advanced Technology R&D Center
Mitsubishi Electric Corporation
Amagasaki, Japan
先端技術総合研究所
三菱電気株式会社
尼崎市兵庫県

13th December 2005

Examiner:
Associate Professor Vladimir Vlassov
Laboratory for Electronic and Computer Systems
Department of Microelectronics and Information Technology
Royal Institute of Technology
Stockholm, Sweden

Industry supervisor:
Terumasa Yasui
Asset Management Systems Group
Advanced Technology R&D Center
Mitsubishi Electric Corporation
Amagasaki, Japan

Abstract

Wireless sensor networks (sensor networks for short) has recently gained large popularity in both academy and industry. These networks of small, often battery powered, sensors can be placed where wired infrastructure is too expensive or impossible to deploy. A sensor network typically consists of a lot of nodes, some nodes might be more advanced than others.

This thesis investigates algorithms for optimizing redundant placement of router nodes in a sensor network in an efficient and reasonable fast way.

There are a couple of requirements on the algorithms for placement of router nodes. Every sensor node in the network must be able to communicate with a predefined computer connected (gateway) node. The communication channel must be redundant, that is if one router in the network breaks or runs out of power, all sensor nodes must still be able to communicate with the gateway node. These requirements should be realized with as few routers as possible.

Sammanfattning

Nätverk av trådlösa sensorer har på senare tid vunnit stort intresse både i industrin och i den akademiska världen. Dessa nätverk av små, ofta batteridrivna, sensorer kan placers på platser där det är för dyrt eller omöjligt att placera kabelanslutna sensorer. Ett nätverk består vanligtvis av många noder, vissa noder kan vara mer avancerade än andra.

Denna rapport undersöker algoritmer för optimering av placering av routernoder i ett trådlöst senor nätverk med redundans på ett effektivt och snabbt sätt.

Det finns ett antal krav på placeringsalgoritmerna. Alla sensornoder i nätverket måste kunna kommunicera med en förutbestämd nod ansluten till en dator (gateway). Kommunikationskanalen måste vara redundant, dvs om en router i nätverket går sönder eller får slut på batterier, måste alla sensornoder fortfarande kunna kommunicera med gateway-noden. Detta ska uppnås med så få routrar som möjligt.

Acknowledgments

This report is a result of my work at Mitsubishi Electric Corporation in Amagasaki, Japan.

I want to express my sincere gratitude to the following people:

Professor Funakubo for helping me find this project.

Atsushi Morita for accepting me to Mitshubishi Electric.

Keiichi Shiotani and **Tadashi Ohi** for accepting me to the Asset Management Project.

Terumasa Yasui for supervising me at Mitsubishi Electric and for helping me during the project.

Takaya Yamamoto for helping me during the project.

The Nogami Family for allowing me into their home on several occasions and letting me live with them after finishing my work at Mitsubishi Electric.

Vladimir Vlassov for supervising me at KTH and helping me make this thesis meet academic standards.

Many thanks goes to all colleagues in the Asset Management Project, my friends at the company dormitory and my friends at Nihongo Salon for making my stay very fun.

I would also like to thank my family and my friends in Sweden for encouraging me before and during this project.

Contents

List of Algorithms

List of Figures

List of Tables

1. Introduction

Wireless sensor networks (often called sensor networks for short) has recently gained large popularity in both academy and industry. These networks of small, often battery powered, sensors can be placed where wired sensor infrastructure is too expensive or impossible to deploy. A network typically consists of a lot of cheap nodes, some nodes might be more advanced than others.

1.1 Goals and Expected Results

The network considered in this thesis has two types of nodes: RFD (Reduced Function Device) and FFD (Full-Function Device). FFDs can communicate with all nodes and route messages through the network, the RFDs can only communicate with FFDs.

A FFD need more memory and more computing performance than a RFD in order to implement the network protocol. Therefore a FFD will consume more energy than a RFD but it acts as a router node. FFDs can form two kinds of network topologies, star and mesh.

The main objective of this work is to find a way of finding good enough placements of as few FFDs (router nodes) as possible, while still fulfilling these requirements:

- All nodes in the network must be able to access a certain (predefined) FFD connected to a computer, a gateway node.
- There must be a configurable least number of possible routes from every sensor node to the gateway.
- Routers cannot be placed anywhere, some placements are impossible.

The expected results of this project is to find a good enough and fast algorithm for placement of routers. And to implement this algorithm in a preferably platform independent and easy to use application.

The algorithms presented are to be evaluated and compared to other existing algorithms for placement of routers.

1.2 Problem Solving Approach

First of all, do a thorough literature study to find out what has already been done in this field of study. This involves reading relevant papers, articles and other documentation.

To make the problem more easy to solve, the problem is considered and solved first in two dimensions. After the two dimensional solution has been evaluated, the three dimensional problem is considered using experience from the two dimensional solution. Yet another way of making the issue easier to handle is to initially consider a non redundant solution.

When writing the applications, the algorithmic part should be easy to exchange for another. That is, the application should be designed with fixed interfaces between the different parts in order to be reusable for evaluating and possibly using several different algorithms.

When the implementation stage is finished, the implementations should be evaluated and tested so that they fulfill the requirements set in Section 1.1. Testing the implementations will most likely require the writing of a application specific simulator application.

1.3 Structure of the Thesis

This thesis is structured as follows:

In Chapter 2 the following is presented: an introduction to wireless sensor networks, previous findings on the subject, more detailed information on the specific issue and why it is important to solve.

Chapter 3 discusses different methods to solve the problem and the design and analysis of the algorithms is done here.

The implementation of the algorithms in an application is described in Chapter 4.

Chapter 5 describes how the evaluation is done and contains the results from the evaluation of the implementations of algorithms. This chapter also analyses the evaluation results.

Chapter 6 summarizes the results of the work, what was done, what more can be done and what could have been done better.

2. Background

2.1 Business Background

It is important for asset management to know the condition of its facilities. With good knowledge of the condition of ones facilities it is easier to plan maintenance, repairs, and replacements.

To find out the condition of a specific facility we need a way to monitor it. Monitoring can for example be done by installing sensor nodes in the target devices, buildings, and other assets. These sensor nodes can continuously collect in depth status all the time in an automatic fashion.

To collect the data from the sensor nodes, we need some kind of network. In a legacy monitoring system, each sensor is connected via cables to a controlling system one by one. As a result, there are at least as many cables as sensor nodes and the cost of installation and fault-finding becomes high.

When trying to find new means of data collection, the cost of deployment must not exceed the cost of deploying a legacy data collection system, it should also have an advantage over other data collection systems available on the market, such as lower price or better ease of deployment. Therefore we need to find means of data collection that are both inexpensive and easy to deploy.

Today there are a couple of ways of reducing the amount of cabling. Most notable are field bus systems, like lonworks, and profi-bus. These two systems use a common bus for communication, which eliminates the need for cables to every sensor from the control system. This reduces the cost, but it still needs cabling for communication and power supply.

I order to further reduce the cost of deployment, a different approach is needed.

2.2 Wireless Sensor Networks

Wireless sensor networks is a relatively new class of networks. Recent advances in techniques for wireless networking and electronics has made it possible to build cheap and power efficient wireless sensor nodes. These nodes can be deployed in large numbers or can be used where traditional wired sensors are too expensive, difficult or impossible to deploy.

Currently, the main purposes for these kinds of networks is expected to be monitoring and control of for example a home or a building.

One key feature in most sensor networks is a self organizing ad-hoc network architecture. This means that the network automatically adds new units as they appear and likewise remove non operational units. This makes the network robust and fault tolerant, providing that there are redundant nodes available.

The network topology in sensor networks currently being developed is based on either WPAN (Wireless Personal Area Networks) or on WLAN (Wireless Local Area Network) technology, although WLAN is mostly considered being too complex and power hungry for these applications.

In [17], Porcino and Wirt discuss the characteristics several standards for the physical and MAC layer in a WPAN. Table 2.1 summarizes and compares the different characteristics of the popular standards for WPANs.

Name	**Max. nodes**	**Max. distance**	**Max. data rate**	**Typical use**
802.15.1 (Bluetooth)	8	30 m	1 Mb/s	Mobile phones and peripherals
802.15.3 (UWB)	N/A[a]	10 m	200 Mb/s	USB, wireless video
802.15.4 (ZigBee)	2^{16} [b]	30 m	250 kb/s	Industry automation, security systems

Table 2.1: Standards used for the physical layer and the MAC layer in WPANs

[a]No figures available at the moment
[b]2^{16} per network, 2^{64} in total

2.2.1 IEEE 802.15.1 (Bluetooth)

The currently most common standard for WPANs is IEEE 802.15.1, commonly called Bluetooth. It is widely used in mobile phones and devices communicating with mobile phones, such as wireless headsets. It is a mature and well tested technology.

Bluetooth uses the same free frequency band as WLAN (802.11b and 802.11g), 2.4 GHz.

Some Characteristics of Bluetooth are Shown in Table 2.2.

Property	**Range**
Raw Data Rate	1 Mb/s
Range	Up to 30 meters
Nodes per Network	8
Topology	Peer-to-Peer
Frequency Band	2.4 GHz

Table 2.2: Characteristics of Bluetooth

According to the Bluetooth Special Interest Group [1] Bluetooth is promoted and used by several large corporations, such as Ericsson, Intel, Nokia, Microsoft, and Toshiba, to name a few.

Drawbacks with Bluetooth when considering sensor networks is its complex protocol, relatively high energy consumption, and the fact that it only supports 7 nodes per master. The data transfer speed is around 1 Mb/s. [7]

2.2.2 IEEE 802.15.3 (UWB)

IEEE 802.15.3, which often go under the name Ultra Wideband (abbreviated as UWB), is a emerging standard with high data transfer speed (about 200 Mb/s). UWB is intended to be used where high bandwidth is needed, such as wireless USB and Firewire. Some characteristics are shown in Table 2.3.

Property	**Range**
Raw Data Rate	110 Mb/s (10m), 200Mb/s (4m)
Range	Up to 10 meters
Location Awareness	Optional
Topology	Peer-to-Peer
Frequency Band	3.1–10.6 GHz

Table 2.3: Characteristics of UWB

One usage scenario is for example connecting a digital video camera to a computer or a DVD-recorder.

UWB uses a currently uncommon way of RF transmission with very high RF bandwidth and short pulses, typically a few picoseconds, when transmitting data. [17]

Since the standard is still very much in it's development, there does not exist any commercially available devices nor any evaluation kits. Specifications on this kind of networks are at the time of writing very brief.

2.2.3 IEEE 802.15.4 (ZigBee)

IEEE 802.15.4, which is used in systems developed by the ZigBee Alliance [19], is a low transfer rate type of network. The transfer rate is 20-250 kb/s depending on the choice of frequency band. This type of networks are sometimes called Low Rate WPAN (LR-WPAN).

This type of network is constructed to work with small resources, slow CPU, small memory, and small amount of battery power.

For the physical layer, there are three possible frequency bands, 2.4 GHz, 868 MHz (For use in Europe), and 915 MHz (for use in the United States). The 2.4 GHz band allows a bit rate of 250 kb/s and has 16 channels. In the 915 MHz band there is 10 channels each allowing 40 kb/s. And the 868 MHz band allows 20 kb/s in one channel. The characteristics of the ZigBee network are summarized in Table 2.4. [10] [2] [6]

Property	**Range**
Raw Data Rate	2–250 kb/s
Range	Up to 10 meters or up to 100 meters with speed trade-offs
Battery Life	Application dependent. Typically optimized for low power consumption. Asymmetrical power consumption. Battery life might be as long as the battery's shelf life.
Latency	10–50 ms or larger than 1 s
Location Awareness	Optional
Nodes Per Network	2^{16}
Topology	Star or Mesh
Complexity	Lower than Bluetooth and WLAN
Types of Traffic	Asynchronous data-centric, optionally synchronous
Frequency Band	2.4 GHz, 868 MHz (Europe) and 915 MHz (US)

Table 2.4: Characteristics of a ZigBee Network

The MAC layer (medium access control layer) and protocol layer of this network standard is constructed to be as power efficient as possible, using low packet header overhead, and variable size packages. This allows a seldom used unit to operate for up to one year on common batteries, maybe even more. These networks are mainly intended to be used in building monitoring and control or in so called intelligent homes.

Theoretically the network protocol supports up to 2^{64} nodes with 2^{16} nodes per network, this is achieved using two different types of devices: FFD

(Full-Function Device) and RFD (Reduced Function Device). The FFDs can communicate with any node within reach and route messages through the network, while the RFDs can only communicate with a FFD.

A RFD is typically a battery powered sensor, a wireless light switch or a similar device. A RFD is typically a device serving data at certain intervals or doing things on demand.

A FFD, a router node, can also have sensor devices connected, but the difference is that it must be able to route network traffic. A number of FFDs make up the backbone of the network, they can form either a star network or a so called Mesh network (or Peer-to-Peer network). The RFDs connect to the network via a FFD. A FFD needs more power than a sensor and might therefore be connected to mains.

In the case of a mesh network several FFDs within communication range from each other communicate, forming a mesh of communication channels. See Figure 2.1 for examples showing the communication differences of networks with star and mesh topology.

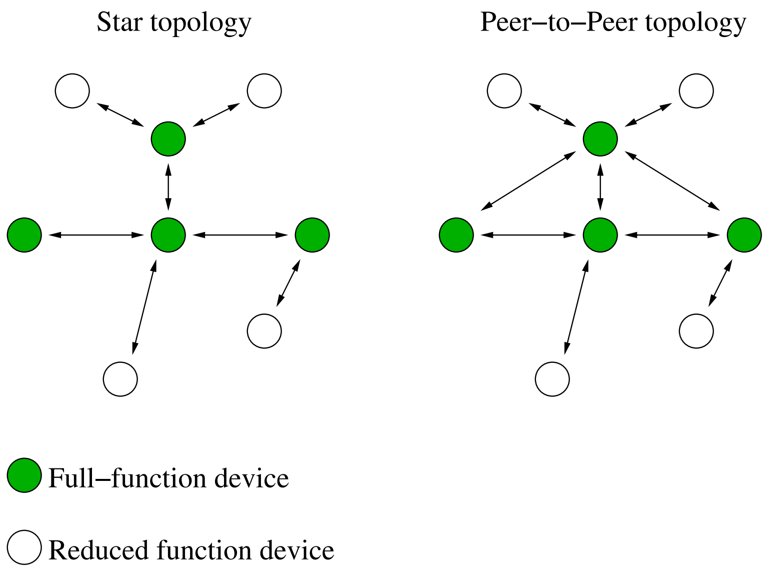


Figure 2.1: Comparison of communication channels in networks with Star and Mesh topology.

For more information on the physical and MAC layer of these networks see the IEEE standard [10] and for the upper layers the ZigBee Alliance's web page [19].

ZigBee is promoted by Mitsubishi Electric, Motorola, Philips, Samsung among others. ZigBee-compliant platforms (the IEEE 802.15.4 radio and the ZigBee stack up to the application layer) are available as either chips or complete modules for use in end products. No end-user products are available at the time of writing.

This thesis will mainly focus on this kind of network, although the results will be applicable on other networks with similar topologies.

2.3 Routing in Wireless Mesh Networks

In the above mentioned (in Section 2.2.3) Mesh topology networks, traffic is routed from the source to the destination via a series of router nodes within radio communication range of each other. These router nodes or other FFDs can form either a mesh or a star topology.

A node wanting to send a message contacts the FFD it is associated with and sends the message. The FFD, unless it is the destination node, forwards the message closer towards the destination. In Figure 2.2 an example of a node sending a message to the gateway with two possible routes is shown.

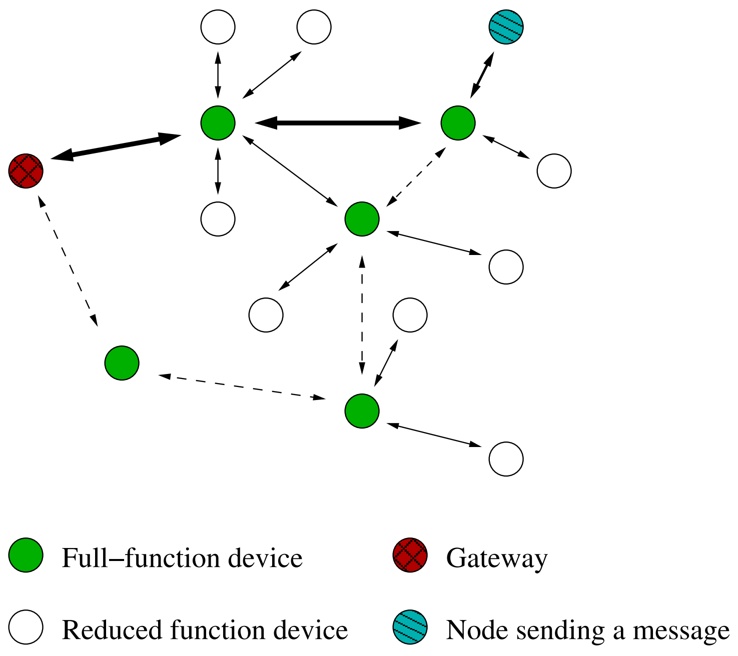


Figure 2.2: A node sending a message to the gateway, with two possible routes.

The biggest difference between wireless sensor networks and most other wireless technologies, is that in common wireless networks, base stations are connected to a network backbone (wired or wireless). This designated network transports data with high bandwidth between base stations and/or to gateways connected to other networks.

In wireless sensor networks, base stations are not linked together with such a designated network, but instead all messages, both those originating at a node and those forwarded, must be forwarded through the network in a peer-to-peer manner. This makes the network more sensitive to congestion and packet loss due to congestion.

Two commonly considered network routing protocols are Ad Hoc On-Demand Distance Vector Routing (AODV) and Dynamic Source Routing (DSR) these protocols are common in mobile ad-hoc networks (MANETs). More details on these routing protocols is available in for example [11] and [16].

More exact details on how the routes are found, issues concerning connecting to the network, recovering from router breakdowns and other data transport issues are beyond the scope of this thesis, that rather focuses on finding optimal placement for the network router nodes.

2.4 Properties of Wireless Sensor Networks

Four very important properties in wireless sensor networks are coverage, connectivity, node lifetime, and reliability.

Coverage is of big interest in other wireless applications than sensor networks as well. Examples are cellular networks for telephones and so called wireless hotspots, where coverage is of great interest.

Connectivity on the other hand, is specific to these kind of ad-hoc networks.

Lifetime is how long time a node can run on its batteries before needing replacement, either by changing batteries or deploying a new node or router.

When combining these properties, we get another important property, reliability. This property is a measurement of how long the network can perform as intended despite node failures, communication interruptions, and other disturbances.

Other quality of service properties worth considering are data throughput and data propagation delay (latency). While still being of big importance to the performance network, these properties are not in the scope of this thesis.

2.4.1 Coverage

Meguerdichian et al. in [15], Huang and Tseng in [9] and Wang et al. in [18] define coverage in homogene networks[1] in similar ways. Their definitions can be summarized as follows:

> A network is fully covered if every location within the area of interest is within sensing range of at least one sensor.

In Figure 2.3 a network consisting of 7 router nodes are covering an area, the covered are is shaded. And In Figure 2.4 one router node is removed leaving a patch in the middle uncovered. Provided that we know that we do not need coverage in the middle, we can conclude that the network still is covered even though we have an uncovered patch.

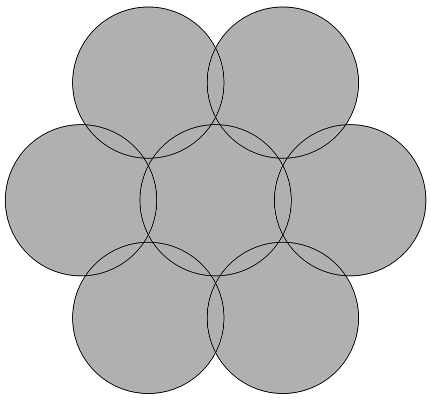


Figure 2.3: Simple Coverage, the shaded area is covered by the network.

If a location is covered by more than one sensor, one can use the notation $k - covered$, where k is the number of sensor nodes monitoring this specific position. In Figure 2.5 three router nodes cover an area and the level of coverage is shown in every subregion.

If we know the area of deployment A_d and the sensing area A_s, we can get an approximate number or a lower bound for the number of nodes needed $n = \frac{A_d}{A_s}$. This lower bound is only valid if the all of the area of deployment needs to be covered.

Coverage is a necessary, but not sufficient property for a working ad-hoc sensor network.

[1]In a homogene network all nodes have the same capabilities.

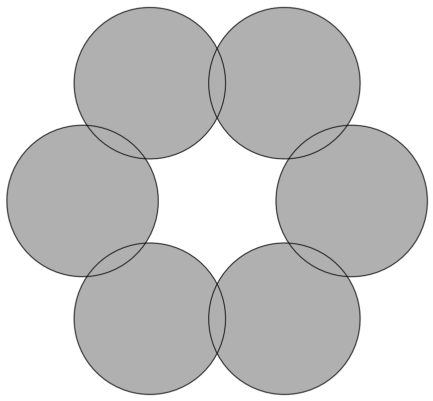


Figure 2.4: The shaded area is covered by the network, leaving a patch in the middle uncovered.

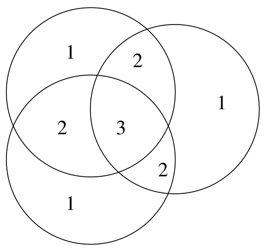


Figure 2.5: k-coverage, the level of coverage is noted in every subregion.

2.4.2 Connectivity

Since wireless sensor networks rely on Peer-to-Peer communication between nodes, the definition of a covered network does not necessarily mean that the network is working and that messages can pass through the network. The network might for example be partitioned but still cover the entire area.

One simple proof of this is if the nodes have the same sensing range as communication range. In this case one can easily construct a network where the entire area is covered without any nodes being able to communicate.

An example of such a network is illustrated in Figure 2.6 where the network is both covered and connected and Figure 2.7 where the network is covered but not connected.

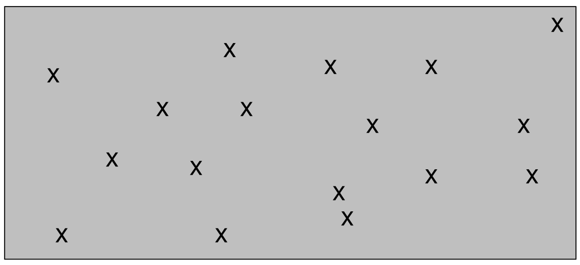


Figure 2.6: Covered and connected network. x represent position where we need network access, the shaded area is covered by the network

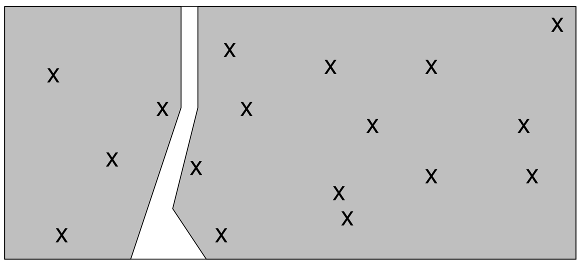


Figure 2.7: Covered but not connected network. x represent position where we need network access, the shaded area is covered by the network

The property whether messages can be transported between nodes is called connectivity. Connectivity is also a necessary, but not sufficient property for a working sensor network.

Providing one know the communication range R_c and the the sensing range R_s of the nodes, it is possible, according to [18], to draw the following conclusion:

A fully covered convex region is connected if $R_c \geq 2R_s$

For proofs and further analysis on this see [18].

2.4.3 Lifetime

The sensor nodes in these networks are usually battery powered. Because of that, they have a limited time of operation. To make the lifetime as long as possible, it is necessary to use low power components when building the sensors and to use a network architecture that allows the sensor to enter some kind of sleep mode for long intervals to save power.

An other way of increasing the lifetime is to deploy more nodes and let them share the workload, two sensor nodes can for example alternately process and send every second sensor reading.

2.4.4 Reliability

The reliability of the network depends on many factors, for example node failure and communication problems.

There is a very simple way of increasing the reliability, deploying redundant nodes. The backside of this is that the cost of deployment increases. But on the other side, the cost of maintenance decreases.

If one has to low redundancy, the network might fail if only one node fails and therefore batteries and failed nodes must be replaced immediately.

By using a higher level of redundancy, we can allow a couple of failed nodes and still know that the network will work as intended. Therefore replacements can be done more seldom.

2.4.5 Application Specific Issues

Because the network considered in this thesis is not populated by homogeneous nodes, certain modifications needs to be made to the definitions of connected and covered networks to make use of results from previous research findings.

Fully covered all sensor nodes in the network has a connection to at least one router node.

Fully connected any randomly selected node in the network must be able to communicate with the gateway node.

What this means is that what others consider as a sensor is a router in this network. Coverage is whether or not the sensor nodes are within communication range of a router. In other words the sensor nodes are monitored by the network of router nodes.

2.5 Related Work

The issue of coverage is not an issue that has arisen with wireless sensor networks, it is of big importance in several other areas.

One example of such an area is cellular networks, an operator wishes to cover as much as possible in order for its customers to be able to make calls. In these kind of network we want as few gaps as possible, so that calls do not get cut off when a caller moves around.

In wireless sensor networks we need to fulfill one more requirement, the network needs to be connected. This makes the problem a bit different from that of cellular networks.

Another more closely related type of networks where coverage and connectivity are considered is ad-hoc networks, often called Mobile Ad Hoc Networks (MANETs). The difference in these technologies is that they usually depend on a "over deployment" of network nodes, that is there are more nodes than necessary in order to achieve coverage and connectivity.

An example of such an ad-hoc network is "The Grid Roofnet" which is described by Chambers in [3]. For redundant mesh networks Hsiao and Kung have some analysis of network designs in [8].

When it comes to optimizing connectivity and coverage in an area where only some parts need coverage there is no previous work done to my knowledge.

As for the routing protocols, a comparison and analysis of AODV and DSR, the two most common network routing protocols for ad-hoc networks, is done by Das et al. in [4].

3. Algorithm Design

This chapter describes the different algorithm approaches used in this project.

To make sensor nodes outside communication range of the gateway node able to send messages, router nodes (FFD nodes) are required. The number of router nodes should be as few as possible, while still letting all sensor nodes communicate with the gateway. It might also be required to have a certain amount of redundancy to allow router failures.

3.1 Placement of Routers

To be able to fulfill the project requirements set in Section 1.1, finding good placements for the network routers is of crucial importance.

First we must see to that all sensors can communicate with the gateway node. When that part is done, we need to add some more routers to get redundancy in case of router failures.

We want to find an optimal solution, giving enough redundancy using as few routers as possible. If too many router nodes are used the redundancy requirement can be fulfilled. But it will be done at a higher cost than necessary because many router nodes are deployed that are not really needed for the required level of redundancy.

3.1.1 Placement Algorithms

According to Huang and Tseng in [9], optimal placement of routers to cover all sensor nodes is a problem similar to the so called *Art Gallery Problem*[1]. This problem has a linear time optimal solution in the two dimensional case, but is NP-hard to solve optimally in three dimensions, according to Marengoni et al. in [13].

Due to the fact that the problem is NP-hard in three dimensions, simplifications has to be made. One can for example try taking advantage of nodes forming clusters, and in other ways preprocess the data to reduce the prob-

[1]The Art Gallery Problem deals with placing as few guards as possible in a polygonal Art Gallery room, while still keeping every point of the gallery under observation by at least one guard.

lem size and thus make it easier to solve. Note that the problem still is NP-hard, but thanks to the smaller problem size it might be solvable.

When the problem is simplified one can try using trivial approaches or develop some heuristics either new or, if applicable, based on the findings in [13] on placement of observers in a three dimensional terrain.

Another possibility is to construct a iterative algorithm that improves the placements of routers in a trivial and unoptimized solution, if such an algorithm is realizable it would enable removal of unnecessary routers.

According to Médard et al. in [14] optimal 2-way redundant routes in connection graphs tend to form rings.

3.1.2 Verification

Verifying that a solution fulfills the coverage requirement can be trivially calculated in $O(n_s n_r)$, where n_s is the number of sensor nodes and n_r is the number of routers. If one uses the findings in [9] the computation can be done in $O(nd \log(d))$, where n is the number of routers and d is the maximum number of routers which intersect any other router's coverage area.

The connectivity and amount of redundancy can be verified and calculated in $O(n_s n_r {n_e}^2)$ using the algorithm described in Section 3.7.3, where n_s, n_r and n_e are the number of sensors, the number of routers, and the number of interconnections (edges in the connectivity graph) respectively.

3.2 Algorithm Overview

3.2.1 Non-redundant solution

Construction of a non-redundant solution can be done in a couple of ways.

The simplest way is to add routers within communication range on a straight line from every sensor to the gateway, excluding sensors within the gateways communication range. This is further discussed in Section 3.3.1.

Another possibly better solution using fewer routers, is to add routers within communication range on a straight line from a sensor to the nearest router or gateway, excluding sensors already within communication range from a router or the gateway as shown in Section 3.3.2.

In order to reduce the problem size, one can try to find optimizations such as sensors forming clusters. This is analyzed in Section 3.3.3.

3.2.2 Redundant solution

When creating a redundant solution, we must first decide what level of redundancy is needed. In some cases it is sufficient with a non-redundant solution, in other cases we might need for example three mutually exclusive routes from every sensor to the gateway. The level of redundancy required is a issue in the specific implementation and is not further discussed in this thesis.

When the required level of redundancy is known, the actual placement is done by first creating a non-redundant solution and then for every level of redundancy running basically the same algorithm as used for non-redundant solutions. This is shown in Section 3.5.

3.3 Non-redundant Router Placement Strategies

Below follows a list of strategies that can be used when constructing non-redundant solutions.

Trivial Router Placement Places router nodes on straight lines from every sensor to the gateway.

Trivial Placement Reusing Already Deployed Router Nodes Places router nodes on straight lines to the closest already deployed router.

Cluster Router Placement In this case the original problem is analyzed and sensor nodes forming clusters are found and are connected as if they where only one sensor.

In the following sections an example setup shown in Figure 3.1 with one gateway and four sensor nodes is used.

3.3.1 Trivial Router Placement

The simplest way of ensuring a connected network is to place router nodes within communication range from each other on straight lines from the gateway to within communication range of each and every sensor node as shown in Figure 3.2.

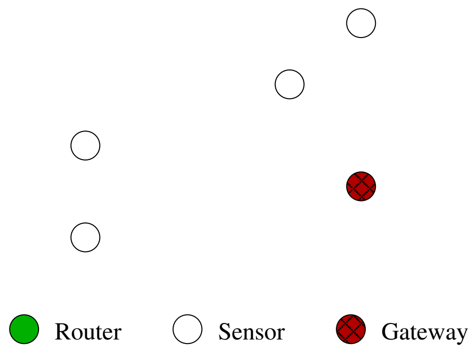


Figure 3.1: Example setup using four sensor nodes and one gateway node.

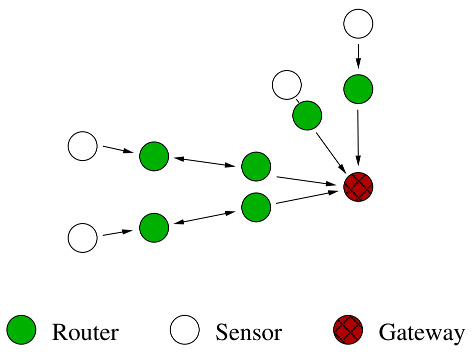


Figure 3.2: Placing router nodes in a trivial way using a exclusive route from every sensor node to the gateway.

This approach results in very many routers, many of them not necessary to obtain a solution where all sensor nodes can communicate with the gateway. In Figure 3.3 where the router-to-router communication range is shown, we can see that either the router marked 2 or the one marked 4 can be removed and the solutions will still satisfy the connectivity and coverage requirements.

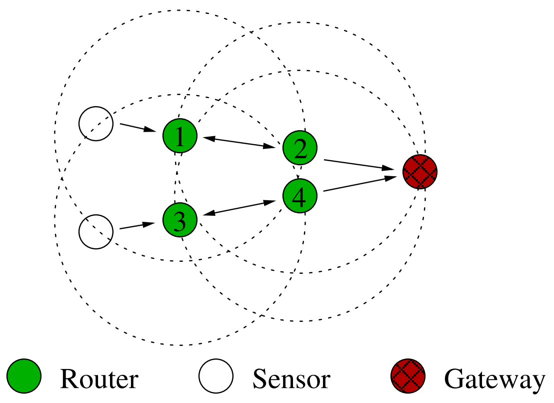


Figure 3.3: In this solution, where the router-to-router communication range is shown with dotted lines, the number of routers is greater than needed and either the router marked 2 or the one marked 4 can be removed.

The algorithm for doing this is fairly straightforward. Unless the sensor node is already within reach of the gateway, place routers along a straight line from the sensor to the gateway. It is given in pseudo code in Algorithm 3.1.

```
Algorithm "Trivial Place Routers"
Input: List of Sensor Nodes $S$, Gateway $g$
Output: List of Router Nodes $R$

for each sensor $s$ in $S$
  if $distance(s, g) > s.range$
    deploy router $r$ at distance $s.range$ from $s$ towards $g$
    add $r$ to $R$
    if $distance(r, g) < r.range$
      continue
    while $distance(r, g) > r.range$
      deploy router $r$ at distance $r.range$ from $r$ towards $g$
      add $r$ to $R$
return $R$
```

Algorithm 3.1: Pseudo code for placement of routers in a trivial way using a exclusive route from every sensor node to the gateway node.

The complexity of this algorithm is $O(n_s d_m)$, where n_s is the number of sensor nodes and d_m is the maximum distance from any sensor node to the gateway.

This expression is obtained by analyzing the algorithm and seeing that for every sensor node, $O(n_s)$, the two coordinates for the placement of a router are calculated and the router is deployed, until the router is within communication range of the gateway, which means $O(d_m)$. Putting the terms together gives the following result $O(n_s d_m)$.

The number of routers placed will also be $O(n_s d_m)$, this is easily verified in the same way as the complexity is calculated.

3.3.2 Trivial Placement Reusing Already Deployed Routes

To reduce the number of unnecessary routers such as those shown in Figure 3.3, one can instead of connecting to the gateway, add routers to connect to the nearest already deployed and connected router. This is illustrated in Figure 3.4.

By deploying routes to every unconnected sensor node using this algorithm, the number of needed routers will be radically reduced in networks with many sensor nodes.

The number of routers needed will asymptotically become $O({d_m}^2)$, as shown below, instead of $O(n_s d_m)$ for the trivial algorithm in Algorithm 3.1. This

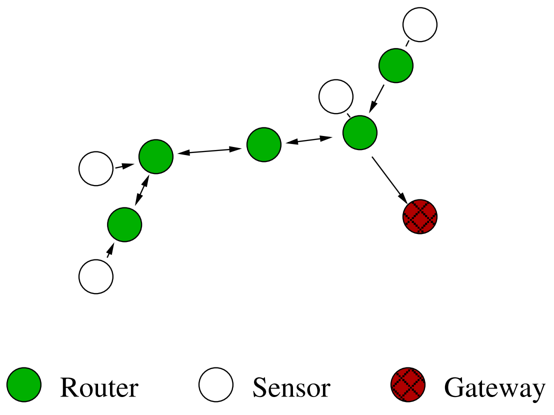


Figure 3.4: Placing routers to connect sensor nodes to the closest already connected router node.

means that if the number of sensor nodes increase in a network but the size is the same, this algorithm will gradually perform better.

When running the algorithm, it first sorts the sensor nodes according to their distance from the gateway. Next step is to connect the sensor node closest to the gateway node with a straight line of as few routers as possible. Then the second closest sensor node is connected to the closest already connected router node or to the gateway node if it is closer. Pseudo code for this algorithm is shown in Algorithm 3.2.

This algorithm is slower than the trivial algorithm in Algorithm 3.1. It requires searching through all already placed routers for every sensor to connect.

The number of routers placed is $O(n_s {d_m}^2)$ when the sensor nodes are few and $O({d_m}^2)$ as the area of deployment becomes saturated, where d_m is the maximum distance from the gateway to any sensor node and n_s is the number of sensors.

The complexity in this case is $O(n_s n_r)$, where n_s is the number of sensor nodes and n_r is the number of router nodes. Using that n_r is $O(n_s {d_m}^2)$ and $O({d_m}^2)$ for the saturated case from above results in $O({n_s}^2 {d_m}^2)$ and $O(n_s {d_m}^2)$ when the area of deployment is saturated with router nodes.

3.3.3 Cluster Router Placement

This method is a way to achieve a linear increase of the performance of the above algorithm in Algorithm 3.2.

```
**Algorithm** "Place Routers Using Deployed Routers"
***Input:*** List of Sensor Nodes $S$, Gateway $g$
***Output:*** List of Router Nodes $R$

for each sensor $s$ in $S$
  if $distance(s, g) > s.range$
    for each router $r_{deployed}$ in $R$
      if $distance(r_{deployed}, s) < distance(r_{selected}, s)$
        $r_{selected} \leftarrow r_{deployed}$
    if $distance(s, r_{selected}) > s.range$
      deploy router $r$ at distance $s.range$ from $s$ towards $r_{selected}$
      add $r$ to $R$
      if $distance(r, r_{selected}) < r.range$
        continue
      while $distance(r, r_{selected}) > r.range$
        deploy router $r_{tmp}$ at distance $r.range$ from $r$ towards $r_{selected}$
        $r \leftarrow r_{tmp}$
        add $r$ to $R$
return $R$
```

Algorithm 3.2: Pseudo code for placement of routers reusing already deployed routers.

The basic idea is to find groups of sensor nodes and connect a whole group of them at one time, reducing the need to search for the closest router node and therefore the computation time needed.

The desired result from this computation is shown in Figure 3.5. Nodes close to each other are groped and all connected using the same route.

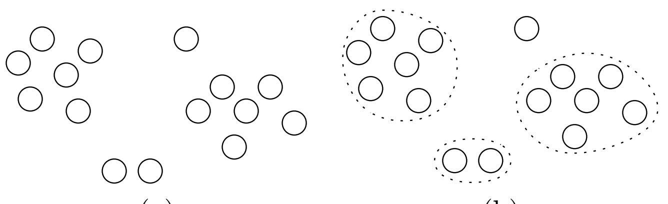


Figure 3.5: (a) shows the sensor placements, in (b) the clusters are marked by the dotted lines

This is done by, for every sensor node s, search for the three closest neighbors and add them to the same cluster as sensor node s if they are within a predefined distance from the node. If a sensor node is part of more than one cluster, the clusters are merged into one.

This algorithm is a modification of the algorithm described by Karypis et al. in [12]. Pseudo code for the algorithm is shown in Algorithm 3.3.

```
**Algorithm** "Find Clusters of Nodes"
Input: List of Sensor Nodes $S$
Output: List of Clusters $C$

for each sensor $s$ in $S$
  for each sensor $n$ in $S$
    if $distance(s, n) < s.range$
      add $n$ to $s.neighbors$ if closer than any other neighbor
        (replace the furthest neighbor if more than 3 neighbors)
  for each neighbor $n$ in $s.neighbors$
    add $s$ to $n.otherneighbors$

$clusteridid \leftarrow 0$ for each sensor $s$ in $S$
  if $s.clusterid = 0$
    $clusterid \leftarrow clusterid + 1$
    $c \leftarrow savetree(s, clusterid)$ (see Algorithm 3.4)
    add $c$ to $C$
return $C$
```

Algorithm 3.3: Pseudo code for finding clusters of sensor nodes.

```
Algorithm "Save Tree"
Input: Node $n$
Output: List of Nodes Belonging to the Same Tree $l$

if not $n.marked$
  set $n.marked$
  add $n$ to $l$
  for each neighbor $s$ in $n.neighbors$
    add $savetree(s)$ to $l$
  for each neighbor $s$ in $n.otherneighbors$
    add $savetree(s)$ to $l$
  return $l$
```

Algorithm 3.4: Pseudo code for saving of neighbor trees.

When the sensor nodes are grouped together the clusters are covered by adding router nodes so that all sensor nodes are within communication range of a router.

The next step is to connect all routers to the gateway. This can be done by using the algorithm shown in Algorithm 3.2 in the previous section substituting the sensor nodes in the placement algorithm for the routers placed by the clustering algorithm.

The complexity for this algorithm is $O({n_s}^2)$, where n_s is the number of sensor nodes. This is easily proved by seeing that there is a nested loop where all sensor nodes are inspected for every sensor node.

If we then use the algorithm from Section 3.3.2 for router placement, we get a complexity of $O(n_c {d_m}^2)$, where n_c is the number of clusters found. From this we can see that the actual complexity will depend on how densely the sensor nodes are deployed.

The number of routers deployed will also highly depend on how densely the sensor nodes are placed. Therefore it is very difficult to give an assessment of the number of routers needed in this solution. Both the time complexity and the number of routers placed will be further analyzed when evaluating the algorithms in Chapter 5.

3.4 Redundant Routes

3.4.1 Background

The solutions that we get using the algorithm presented in Section 3.3.3 and Section 3.3.2 will work fine as long as all router nodes are functional. But if one or more routers fail, large portions of the network might fail.

An example of such a situation is shown in Figure 3.6. Here only one failed router results in the entire network breaking down.

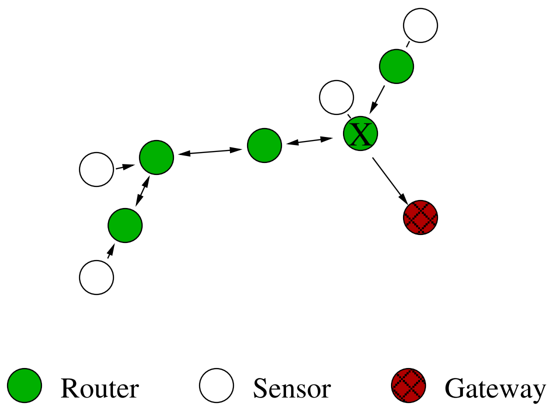


Figure 3.6: One failed router node (marked with an “X”) results in all sensor nodes being unable to communicate with the gateway.

Because of this we need to add redundancy to the network. This means that all sensor nodes need two or more routes to communicate with the gateway in order to ensure a reliable network.

Any two routes or paths through the connection graph that do not share any router nodes are mutually exclusive. The level of redundancy for a sensor node is the number of mutually exclusive routes.

The level of redundancy for the entire solution is the minimum level of redundancy from any sensor node to the gateway.

3.4.2 Trivial Redundancy

The simplest way to achieve a k-redundant solution is to simply deploy k router nodes at every position designated by a non-redundant algorithm.

This solution is simple, fast and straightforward. But in most cases it is possible to solve the problem with fewer routers.

A simple example of such a situation is shown in Figure 3.7. Here 5 router nodes are used for the non-redundant solution, which means 10 router nodes for the 2-redundant solution.

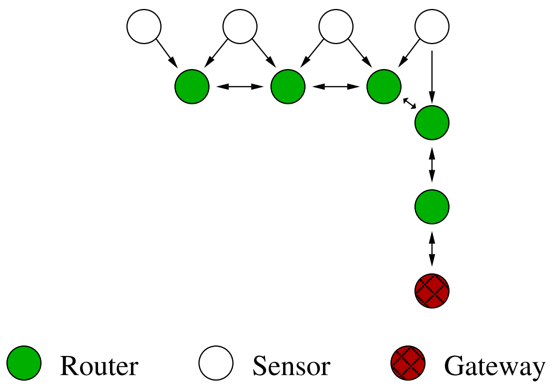


Figure 3.7: Non-optimal redundant solution, all router nodes are simply multiplied.

A slightly better solution of this problem is shown in 3.8. Here 9 routers are used to solve the same problem.

In this example the number of router nodes saved is only one, but in a bigger network more router nodes can be saved.

This result also complies well with what Médard et al. writes in [14], optimal 2-way redundant routes in connection graphs tend to form rings.

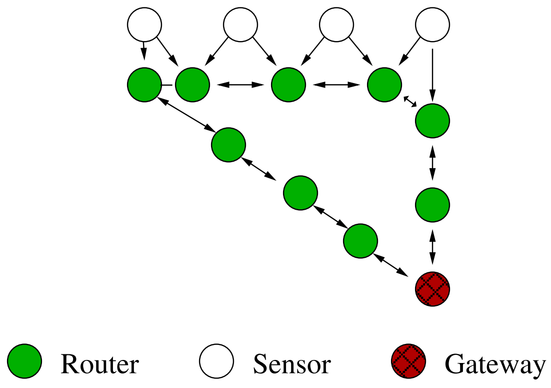


Figure 3.8: Better Redundant Solution

3.5 Non-trivial Redundancy

This algorithm is split into two parts. The main part is the algorithm used for counting the number of mutually exclusive routes in the connection graph. The second part places the router nodes needed.

Another important detail before actually starting creating the redundant routes is to check the connection graph so that all nodes are connected to all router nodes within communication range and any connections missing are added. This is needed because of the design of the non-redundant algorithms presented earlier.

Furthermore, the algorithm for computing mutually exclusive routes is divided into two steps, the first step is a transformation of the bidirectional connection graph into a unidirectional connection graph. The second step computes the maximum flow from a sensor node to the gateway node.

By defining a certain maximum flow on certain connections in the graph, the maximum flow from the sensor to the gateway will be the same as the number of mutually exclusive routes. This is further explained below.

If we use this algorithm to check every sensor's number of mutually exclusive connection to the gateway before making a new connection, we can avoid constructing unnecessary routes.

The result after using the following algorithm with a desired level of redundancy of 2 is as shown in Figure 3.9.

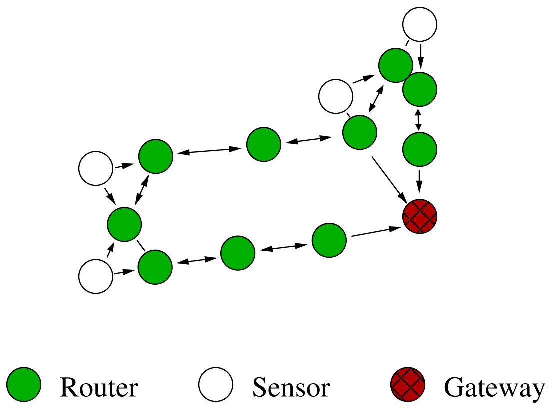


Figure 3.9: Every sensor has two mutually exclusive routes to the gateway.

3.5.1 Counting Number of Mutually Exclusive Routes

This is a computationally difficult problem. When counting number of routes in a graph with few connections, as in Figure 3.10, it is simple to count the number of mutually exclusive routes.

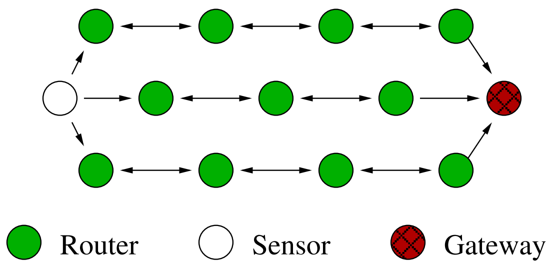


Figure 3.10: In a simple graph with few connections, counting number of routes is simple.

When the graph becomes more complex with many connections, as in Figure 3.11, counting the number of mutually exclusive routes becomes hard.

3.5.2 Transformation of the Connection Graph

The transformation from a graph with bidirectional edges to a graph with unidirectional edges is done by substituting every router in the graph for two routers connected together with a one way connection.

The idea is to let all incoming edges go to one side of the router compound and to let all outgoing edges start from the other side, and thus get a single

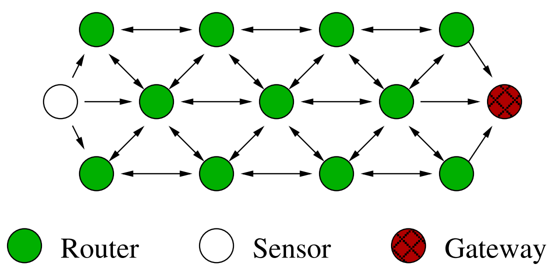


Figure 3.11: In a more complex graph with many connections, counting number of routes is quite difficult.

edge where all communication through the original router has to go. Figure 3.12 shows an example of how a router is substituted.

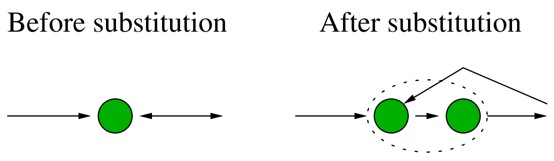


Figure 3.12: Substitution of a single router in a connection graph.

In Figure 3.13 a small graph with bidirectional edges is transformed to a graph with only unidirectional edges.

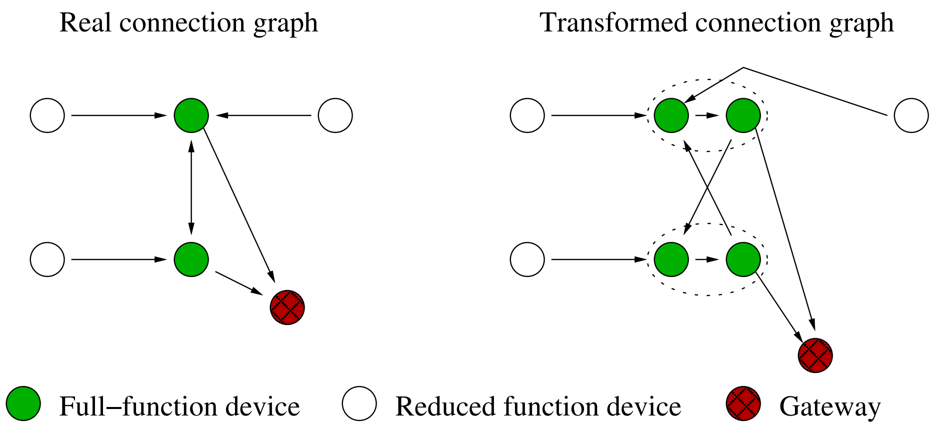


Figure 3.13: Transformation of a graph with bidirectional edges to an unidirectional graph.

Every connection graph has one and only one transformation, that is the transformation is unique and it does not depend on the order of traversal.

3.5.3 Computing Maximum Flow in the Connection Graph

After the graph transformation, the graph has only unidirectional connections. In such a graph it is possible to calculate maximum flow between any two nodes.

To compute the number of mutually exclusive routes from a sensor node to the gateway node through the graph, the maximum flow through every connection is set to 1. This means that every connection can accommodate one and only one route.

In this weighted directed graph it is possible to compute the maximum flow from a source to a sink using The Ford-Fulkerson Algorithm. In our problem, the weights are the same as the maximum flow through the respective edges, the source is a sensor node and the sink is the gateway.

The Ford-Fulkerson Algorithm ([5], Section 8.2.2) depends on finding paths where more flow can be pushed through, so called augmenting paths. Forward edges using less than full capacity and backward edges with flow more than 0 can be used as a path. The graph is traversed in a breadth first manner, searching for unused paths in the graph.

Since the maximum flow through any edge in our problem is 1 and it is either used or not, the algorithm can be simplified. The flow is computed as follows.

1 Find usable path through the graph. Either by using a unused path or by using a used edge in the reverse direction.

2 Mark path as used, edges used in the reverse direction are "unused".

3 Restart from 1 until there are no more paths available.

4 Sum the flows in the edges that start at the source to get the total flow through the graph.

In Figure 3.14 an example of finding and using paths is shown. In this example a path using flow 1 is found and marked as used. Then a second path is found by using one of the edges backwards. When the second path is marked as used, the edge used backwards its flow is reduced to 0.

Algorithm 3.5 shows the algorithm for finding augmenting paths. ([5], Section 8.2.1)

An outline of the remaining part of the algorithm, the part used when marking paths as used and summing up the total flow, is shown in Algorithm 3.6.

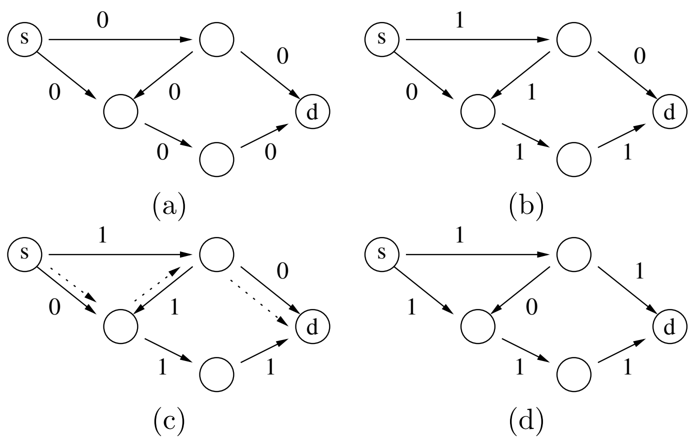


Figure 3.14: Finding augmenting paths in a graph: In (a) the graph has no flow, (b) shows the result when one route is used, in (c) a new possible route using a backward edge is indicated with dashed lines, (d) is the final result with a total flow of 2.

```
**Algorithm** Find Augmenting Path
***Input:*** weighted directed graph $G$, source $s$, destination $d$
***Output:*** path $\pi$

for each node $n$ in $G$
  $n.visited \leftarrow false$
create FIFO-queue $Q$
put $s$ into $Q$
while $Q$ is not empty
  get node $n$ from $Q$
  if $n = d$
    build path $\pi$ recursively starting at n.route
    return path $\pi$
  $n.visited \leftarrow true$
  for each edge $e$ to and from $n$
    if not $e.destination.visited$ and $e$ is forward edge and $e.flow = 0$
      $e.destination.route \leftarrow e$
      put $e.destination$ into $Q$
    if not $e.source.visited$ and $e$ is backward edge and $e.flow = 1$
      $e.source.route \leftarrow e$
      put $e.source$ into $Q$
```

Algorithm 3.5: Pseudo Code for the Algorithm used to find augmenting paths

In [5], Section 8.2, Goodrich and Tamassia provide a more detailed description and an analysis of the Ford-Fulkerson Algorithm.

```
**Algorithm** Maximum Flow
Input: weighted directed graph $G$, source $s$, destination $d$
Output: maximum flow $maxFlow$ from $s$ to $d$

for each edge $e$ in $G$
  $e.flow \leftarrow 0$
$maxFlow \leftarrow 0$
repeat
  traverse $G$ starting at $s$ to find a augmenting path $\pi$ to $d$, see Algorithm 3.5
  if a path $\pi$ exists then
    $maxFlow \leftarrow maxFlow + 1$
    for each edge $e$ in $\pi$ do
      if $e$ is forward edge then
        $e.flow \leftarrow 1$
      else
        $e.flow \leftarrow 0$
  else
    stop
```

Algorithm 3.6: Pseudo Code for the Simplified Ford-Fulkerson Algorithm

3.5.4 Placing Routers in the Redundant Solution

Before creating any redundant routes from a sensor node, the number of routes in the graph from the sensor to the gateway is computed using the above discussed algorithm. If the number of connections already is sufficient, no action is taken. In the case that more connections are needed, new independent routes are created with the algorithm in Algorithm 3.7.

The basic ideas behind this algorithm is to first check whether the sensor nodes already have enough connections to satisfy the redundancy required. This computation is done by using the "Maximum Flow" algorithm shown in Algorithm 3.6.

Secondly the connections needed are added. The sensor nodes are examined starting with the node furthest away from the gateway and if a node does not have enough connections, a new connection is made.

This new connection is made by searching for the closest router node that is on another branch in the connection graph than the sensor node itself.

```
Algorithm Create Redundant Route from $s$ to $d$
Input: connection graph $G$, source $s$, destination $d$, gateway $g$
Output: connection graph $G$

$R \leftarrow$ router nodes in $G$
$S \leftarrow$ sensor nodes in $G$
sort $S$ by decreasing distance to $g$
for ($i = 0$; $i <$ desired level of redundancy; $i++$)
    for each sensor $s$ in $S$
        compute number of routes $n$ from $s$ to $g$ in $G$ using the "Maximum Flow" algorithm in Algorithm 3.6
        if $n \leq$ desired redundancy
            for each router $r$ in $R$
                if $r$ is not on same branch in the graph as $s$
                    build route from $s$ to $r$, this can be done in the same way as in any of the non-redundant algorithms
                    continue
```
Algorithm 3.7: Pseudo Code for Creating of Redundant Routes

Branches in the graph is illustrated in Figure 3.15. In this figure the node marked with X belongs to one branch and the node marked with Y belongs to another branch, while Y and Z belongs to the same branch. One can see that the only node X and node Y share is the gateway node.

The actual placement is then done in the same way as in the non-redundant algorithms. This is repeated for every sensor node and every level of redundancy needed.

3.5.5 Algorithm Analysis

When inspecting Algorithm 3.7 we see that the complexity depends on several factors. At first we have a loop counting the redundancy, then we iterate through the sensor nodes and finally we compute the maximum flow. After the flow has been computed, the already placed routers are iterated and searched for one on a different branch.

The complexity for the flow computation also depends on several factors making it difficult to state. If we first analyze the algorithm for finding augmenting paths, Algorithm 3.5, we find that it depends on the number of edges or connections in the graph n_e. Making up a complexity of $O(n_e)$. This is more rigorously analyzed by Goodrich and Tamassia in [5], Section 8.3.

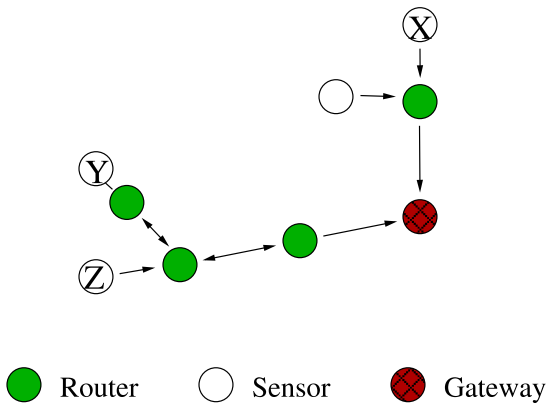


Figure 3.15: Branches in a Connection Graph. The node marked with X and the node marked with Y belong to different branches, while Y and Z belong to the same branch.

When analyzing Algorithm 3.6 we find that the loop runs until all paths are found, that is the algorithm complexity depends on the total flow f. It also marks nodes, this can in the worst case include all router nodes, n_r.

Summarizing these two parts we get a complexity of $O(fn_en_r)$ for computing the maximum flow.

The branch computation has complexity $O({n_h}^2)$, with n_h the maximum number of hops in the graph. Since the number of edges n_e and number of routers n_r always is greater or equal to the maximum hop count n_h the flow computation is the dominant part.

The placement of routers, using a similar algorithm as the trivial one in Section 3.3.1 will need $O(d)$, where d is the maximum distance between any two nodes in the graph. Since $O(fn_en_r)$ is a larger factor than $O(d)$, this factor can be ignored.

Combining the results above yields a total complexity for making redundant routes of $O(rn_s(fn_en_r)^r)$, with r being the level of redundancy requested. This means that the time needed by this algorithm will increase steeply as the level of redundancy increase.

Due to the many factors involved in this algorithm, it is not practical to give an assessment of the number of routers placed. Instead this is left to Chapter 5, where experimental results are presented.

3.6 Optimization

When routers are placed using the algorithms above, some router nodes not needed to achieve a connected and covered solution with the redundancy required might be deployed. This is unwanted and because of this problem, some kind of optimization is needed.

3.6.1 Intermediate Optimization

After deploying routers for non-redundant connectivity, some routes might be longer than necessary and some routers nodes might not be needed to have a connected and covered network.

To solve this problem, all sensor nodes are reconnected to a reachable router node with as short route to the gateway as possible. The same is done with every router node. Router nodes with only connection to another router node are removed. This is repeated until no more routers can be removed.

3.6.2 Final Optimization

When the routes are created using the redundant algorithm in Section 3.5, more nodes than necessary might be deployed. An example of how such a situation can occur is shown in Figure 3.16.

To find the router nodes that are removable, all routers are iterated and one by one temporarily removed from the solution. For every removed router, the number of routes is computed by using the algorithm discussed in Section 3.5. If all sensor nodes still have enough redundancy, the router node is permanently removed. The pseudocode for the algorithm is shown in Algorithm 3.8.

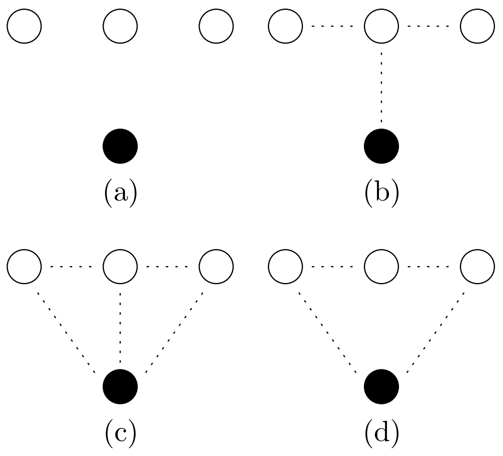


Figure 3.16: Example of how a solution that needs optimization can occur: (a) shows the problem to solve, three sensor nodes an one gateway node. The required redundancy in is 2 in this example. In (b) all sensor nodes are connected to the gateway node without redundancy. (c) shows the solution after the redundant routes are created, the problem now is that the sensor node in the middle has more redundancy than required. In (d) the desired result is shown.

```
**Algorithm** Optimize Connection Graph
***Input:*** connection graph $G$, source $s$, destination $d$, gateway $g$
***Output:*** connection graph $G$

$R \leftarrow$ router nodes in $G$
$S \leftarrow$ sensor nodes in $G$
sort $S$ by decreasing distance to $g$
for each sensor $s$ in $S$
    for each router $r$ in $R$
        mark $r$ as failed
        compute number of routes $n$ from $s$ to $g$ in $G$ using the “Maximum Flow” algorithm from Algorithm 3.6.
        if $n \leq$ desired redundancy
            mark $r$ as non failed
        else
            permanently remove $r$
```

Algorithm 3.8: Pseudo Code for Optimization of the Connection Graph

3.7 Verification

Verifying that the algorithm's chosen router placements fulfill the requirements set in Section 1.1 is very important. Therefore we need a way of verifying the coverage, Connectivity and Redundancy of the solutions.

3.7.1 Coverage Verification

Verification of coverage can be done in a fast enough way using a trivial approach where one simply tests if there are any routers within communication range from the sensor. The complexity for this computation is $O(n_s n_r)$, where n_s is the number of sensor nodes and n_r is the number of router nodes.

3.7.2 Connectivity Verification

Using a trivial breadth first graph search when computing the connectivity is reasonably fast and will require $O(n_s n_r n_e)$, where n_s is the number of sensor nodes, n_r is the number of router nodes and n_e is the number of edges (connections) in the graph.

The graph search should start at the gateway and traverse as much of the graph as possible. If the search reaches all sensor nodes, the connectivity requirement is fulfilled.

3.7.3 Redundancy Verification

To verify the level of redundancy, the "Maximum Flow" algorithm described in Section 3.5 can be used. For every sensor node, the number of routes are computed and if all sensor nodes has enough routes the requirement is fulfilled.

Since redundancy requires both coverage and connectivity, we can verify our solution by only computing the level of redundancy in the solution. If the level of redundancy is 0 for any sensor node, the graph is either not connected or not covered.

4. Implementation

The algorithms presented in Chapter 3 are implemented in an application in order to be able to evaluate the actual time needed to compute a solution.

The application is written in Java, because it allows us to run the application on many different platforms and environments.

In order to keep the application simple and easy to debug, it is split into different parts with fixed interfaces in a Model-View-Control fashion.

The model-part, which is the main part of the application, consists of several Java-classes. It takes care of the actual computation and it is here the algorithms from Chapter 3 are implemented and it is here the problem and solution is stored.

The Java-classes for the user interface, the view, does not know anything about the actual computation done. It simply enables the user to communicate with the model-part and the control-part. It shows the solution and lets the user change the different problem parameters.

When a user wants to solve a problem, the user invokes the control-part via for example a button in the user interface. The Java-classes in the control-part invokes methods in the model that actually do the computation.

A diagram showing the more important classes of the application and how they are related is shown in Figure 4.1

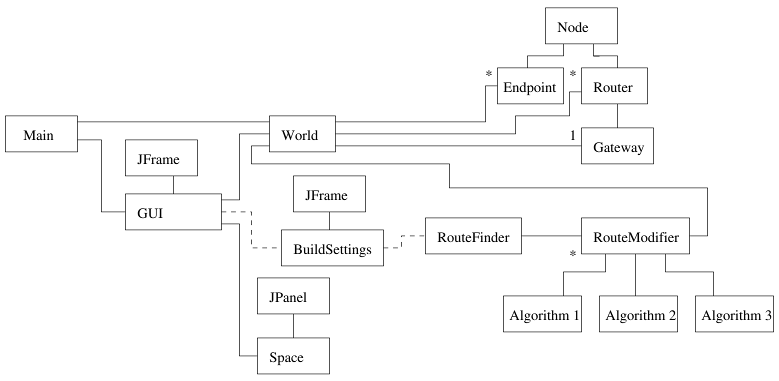


Figure 4.1: Class diagram showing the most important classes in this application. An asterisk means that there are several instances of the class or interface.

Below follows a more detailed description of the classes and interfaces involved.

4.1 View

The view-part of the application consists of a Graphical User Interface, the GUI, Space and BuildSettings classes.

Here the problem and, when computed, the solution is shown. The user interface also has means to adjust problem parameters and a way to select algorithms to run and to start the computation.

Another feature worth noting is the possibility to run several computations on randomly created problem setups.

An example of how such a setup can look like is shown in Figure 4.2, in this example the number of sensor nodes is 100 and the gateway is placed in the middle. In Figure 4.3 the solution for a redundancy of 2 is shown, in this case we needed 163 router nodes to achieve a solution.

To be able to set the desired level of redundancy and to select which algorithms that are to be used to solve the problem, there is a settings dialog. This dialog window is shown in Figure 4.4 and here we can see that the example above was solved by first using the cluster algorithm from Section 3.3.3 named "Clusters" and then using the redundant algorithm from Section 3.5 named "RedundancyMaker". The modifier classes used are explained more in Section 4.2.4.

4.2 Model

This part consist of the World, Endpoint, Gateway, Router, Node and the RouteModifier class. The classes extending the RouteModifier class are also a part of the model.

4.2.1 The World Class

The World class stores the nodes and provides methods used by other classes to get the lists of sensor nodes (endpoints), the list of router nodes and the gateway.

It also provides methods for saving, loading and randomizing setups.

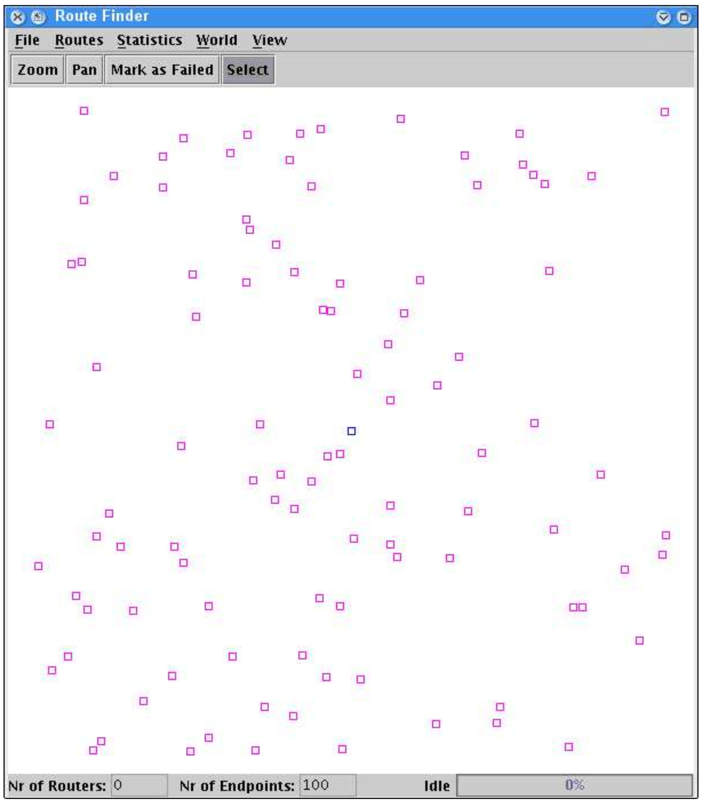
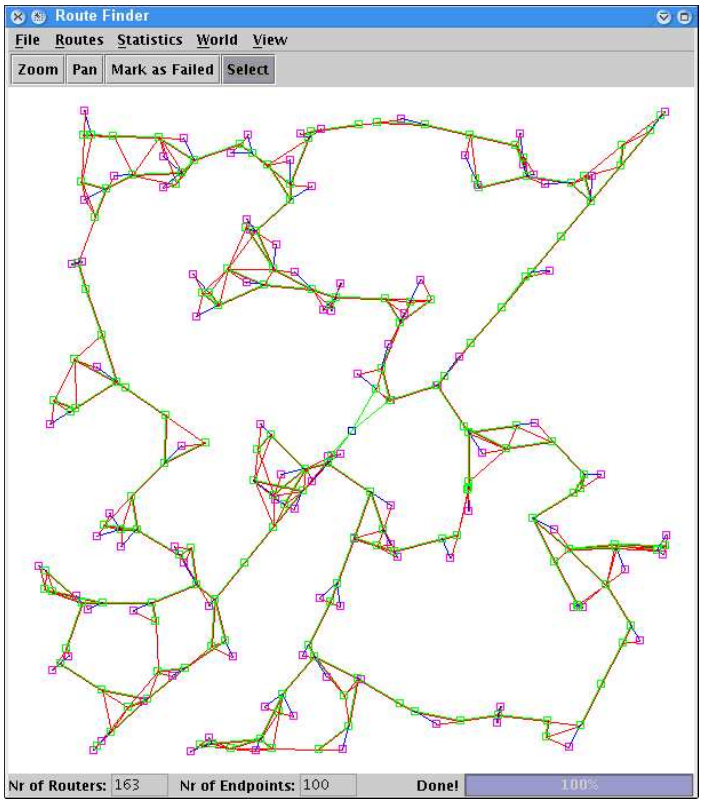
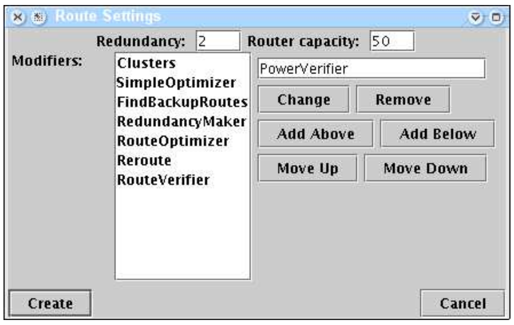


Figure 4.2: Main Window with Problem to Solve. The blue square represents the gateway node and the ones in magenta are the sensor nodes.



Nr of Routers: 163 Nr of Endpoints: 100 Done! 100%

Figure 4.3: Main Window with Solution The blue square represents the gateway node, the ones in magenta are the sensor nodes and the green ones are the router nodes. The connections are marked with lines.



Redundancy: 2 Router capacity: 50

Modifiers:

Clusters
SimpleOptimizer
FindBackupRoutes
RedundancyMaker
RouteOptimizer
Reroute
RouteVerifier

PowerVerifier

Figure 4.4: Route Finder Settings

4.2.2 The Node Classes

The Node Class This class keeps track of all the node properties such as communication ranges. It also stores the connections to other nodes and the number of hops to the gateway.

It provides methods for setting and getting parameters and methods to add connections and to get the list of connections.

The Endpoint Class This class extends the Node class. It sets the default communication range for a sensor node and the node as a sensor

The Router Class This class also extends the Node class. It sets the default communication range for a router node and marks the node as a router.

The Gateway Class This class extends the Router class. It marks the node as a gateway.

4.2.3 The RouteModifier Class

This class exists as a interface between the control part and the model part. It is used by the implementations of the algorithms presented in Chapter 3 and by the RouteFinder class.

4.2.4 Algorithms Extending the RouteModifier Class

All of the algorithms presented in Chapter 3 are implemented using the RouteModifier interface. Below follows a list of the classes and what algorithms they implement.

The Trivial Class Here the trivial algorithm from Section 3.3.1 is implemented.

The TrivialReuse Class In this class the trivial algorithm with reuse, from Section 3.3.2, is implemented.

The Clusters Class This class implements the cluster algorithm from Section 3.3.3.

The RedundancyMaker Class The algorithm from Section 3.5 is implemented here.

The SimpleOptimizer Class Here the optimizer from Section 3.6.1 is implemented.

The RouteOptimizer Class The optimizer from Section 3.6.2 is implemented in this class.

4.3 Control

The control consist of the RouteFinder class. It is invoked from the BuildSettings class and it invokes the RouteModifier classes chosen in the BuildSettings dialog window.

5. Evaluation

Evaluation of the algorithms presented in Section 3 has been done by implementing each one in a computer application as described in Section 4 and performing evaluation experiments using the application. This chapter presents the evaluation experiments and results of the evaluation.

The performance of the algorithms has been evaluated by running the application with random setups with one parameter varied and the other fixed. The number of sensor nodes, router nodes used and the time elapsed is measured for different algorithm combinations.

The parameters varied are the number of sensor nodes and the level of redundancy.

For all of these tests, the router-to-router communication range is set to 30 meters and the sensor-to-router communication range is set to 15 meters. The area of deployment is a square of 400 meters.

This means that we have an area of deployment of $A_d = 160000\ m^2$ and a router to sensor communication area of $A_c = 15^2\pi\ m^2$ for every router node.

Seeing that we fulfill the requirement 2.4.2 stating that the router to router communication range is twice the router to sensor communication range and using the formula from 2.4.1 means that the minimum of router nodes placed in a non-redundant solution filled with sensor nodes is $\frac{A_d}{A_c} \approx 226$.

It should be noted that it is impossible to create a solution with this number of routers. This is because we need some overlap of the communication areas due to the circular shape of the coverage areas.

5.1 Non-redundant Solutions

In this section the algorithms presented in Section 3.3 are evaluated.

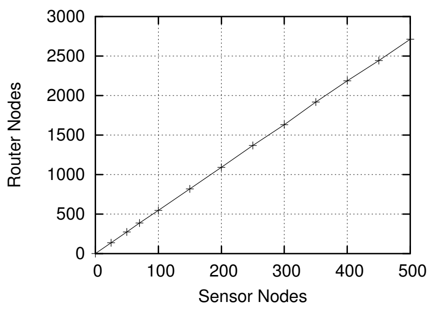
5.1.1 Trivial Algorithm

As stated in Section 3.3.1, the Trivial Algorithm has complexity $O(n_s d_m)$ and the number of router nodes placed is also $O(n_s d_m)$, where n_s is the number of sensors and d_m is the maximum distance from the gateway to the sensor nodes.

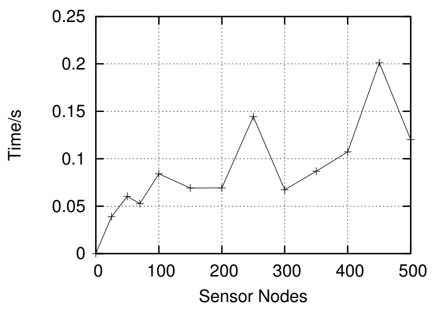
Figure 5.1 shows the number of router nodes plotted against the number of sensor nodes and the computing time measured, when keeping the area of deployment fixed.

In this figure we can clearly see that the number of routers placed is proportional to the number of sensor nodes. Due to the short computing time for this algorithm it is difficult to draw any conclusions from the computing times measured.

We can also see that this algorithm places very many routers, many more than the theoretical minimum of 226 router nodes.



Number of router nodes placed



Time used for computation

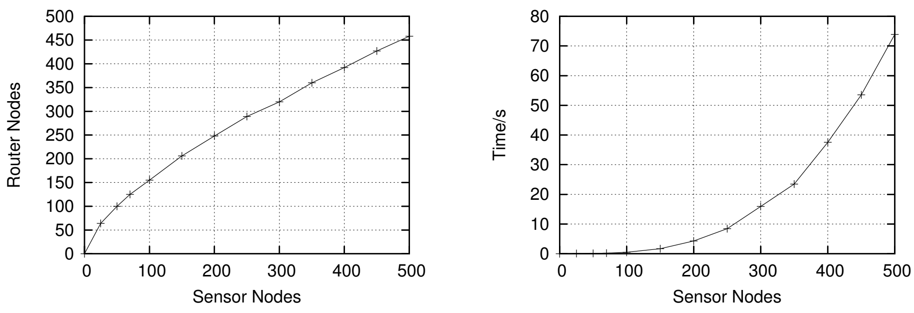
Figure 5.1: Trivial Algorithm

5.1.2 Trivial Algorithm with Reuse

By looking at Figure 5.2, we can see that the number of routers placed by the trivial algorithm with reuse from Section 3.3.2 is less than with the trivial algorithm above. But the time needed for computation is increased.

In the figure we can see that the number of router nodes placed still seems to be linearly related to the number of sensor nodes but that it is so with a lower coefficient. The number of router nodes placed is quite a bit higher than the theoretical lower bound of 226 router nodes.

The time needed is clearly related to the square of the number of sensor nodes.

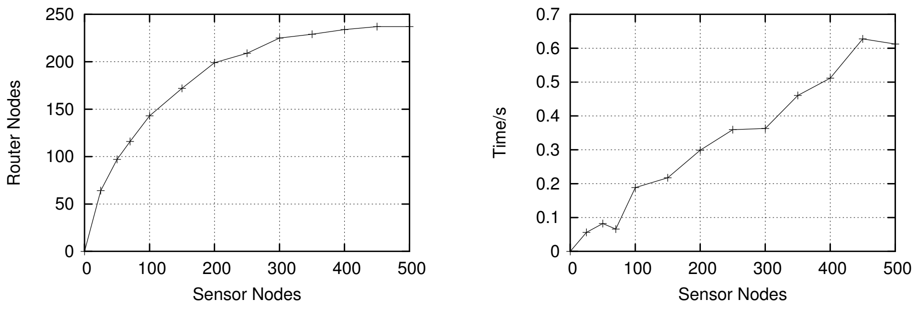


Number of router nodes placed Time used for computation

Figure 5.2: Trivial Reuse Algorithm

5.1.3 Cluster Algorithm

Figure 5.3 shows the number of router nodes placed and the time needed by the cluster algorithm from Section 3.3.3.



Number of router nodes placed Time used for computation

Figure 5.3: Cluster Algorithm

By studying this figure, we can see that this algorithm has a asymptotic behavior as the number of sensor nodes increase. It can be shown that the maximum depends on the size of the area of deployment.

This result complies well with the theoretical result of approximately 226 router nodes obtained in the beginning of this chapter.

The time needed seems to be linearly related to the number of sensor nodes.

5.1.4 Summary

Figure 5.4 shows a comparison of number of routers between the algorithms. Here we can clearly see that the trivial algorithm has the highest number of router nodes placed.

We can see that the trivial algorithm with reuse solves the problem with less than one fifth of the router nodes needed by the trivial algorithm.

It also shows that the cluster algorithm needs even fewer nodes making it the best choice.

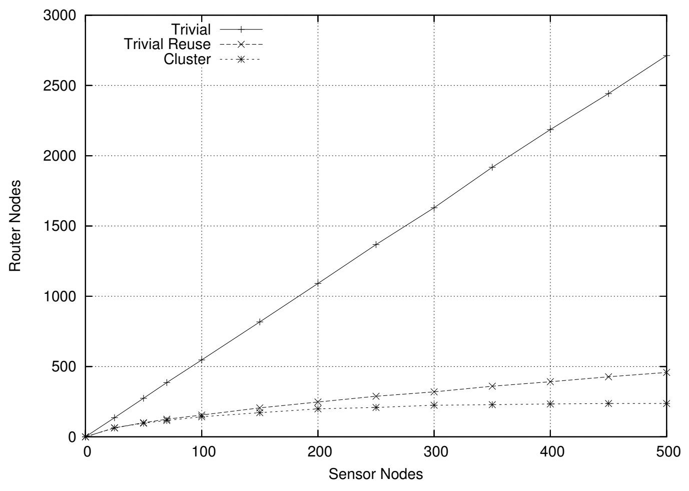


Figure 5.4: Number of router nodes placed by the different algorithms.

In Figure 5.5 a comparison of the time needed by the algorithms is shown. Here we can see that the trivial algorithm with reuse needs the most time and that the trivial algorithm is the fastest one, placing the cluster algorithm in second place for most problem sizes.

Figure 5.6 illustrates how the number of router nodes per sensor node changes as the number of sensor nodes increase.

We can see that the "Trivial" algorithm has a fixed number of router nodes per sensor node, not depending on the number of sensor nodes. Both the "Trivial Reuse" and the "Cluster" algorithms need fewer router nodes per sensor node as the number of sensor nodes increase.

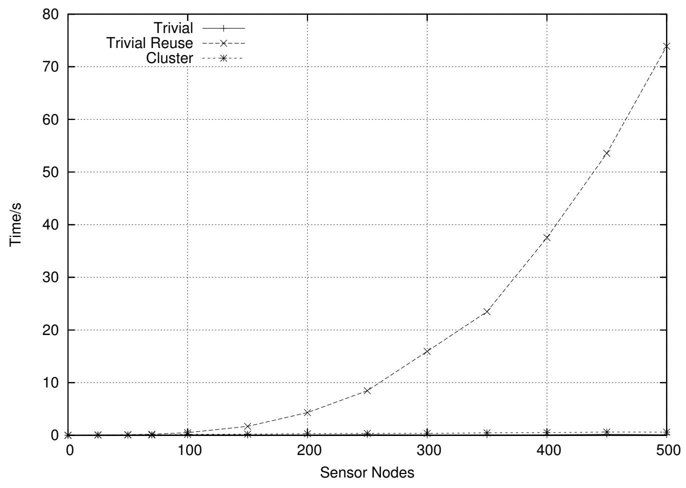


Figure 5.5: Computation time needed by the different algorithms.

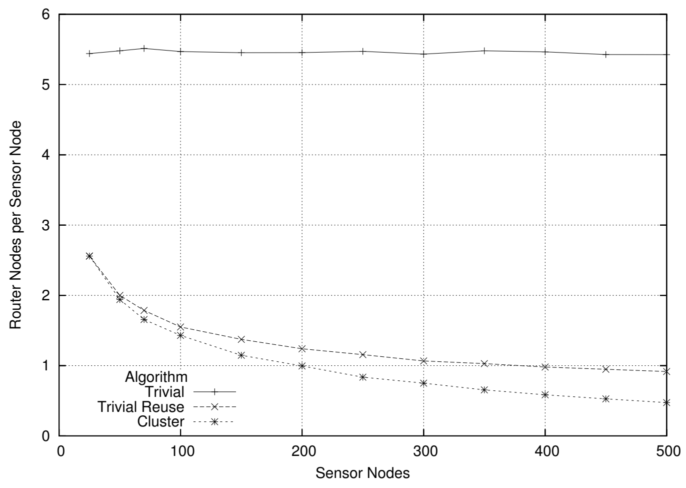


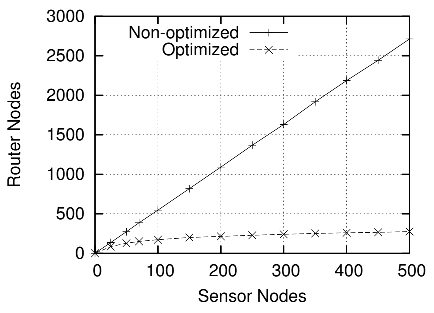
Figure 5.6: Number of Router Nodes per Sensor Node. Non-redundant and Non-optimized solution.

5.2 Non-redundant Optimized Solutions

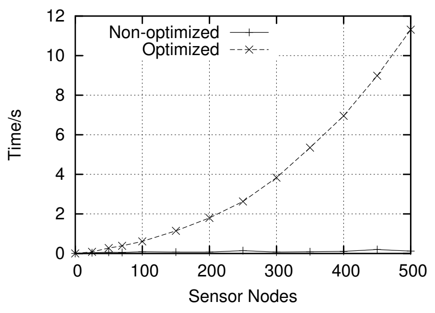
In this section the algorithms presented in Section 3.3 optimized by the optimization discussed in Section 3.6.1 are evaluated.

5.2.1 Trivial Algorithm

In Figure 5.7 we can see that the number of router nodes can be radically reduced in the trivial solution, but on the other hand it takes quite a lot more time to reach a solution.



Number of router nodes placed

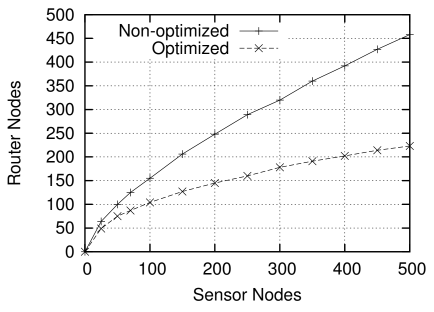


Time used for computation

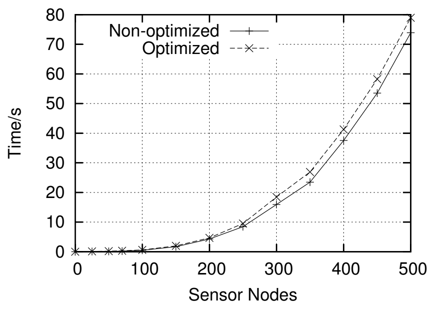
Figure 5.7: Trivial Algorithm with Optimization

5.2.2 Trivial Algorithm with Reuse

With the trivial algorithm with reuse, Figure 5.8, the reduction is still quite big with the cost being that the time needed is increased a bit.



Number of router nodes placed

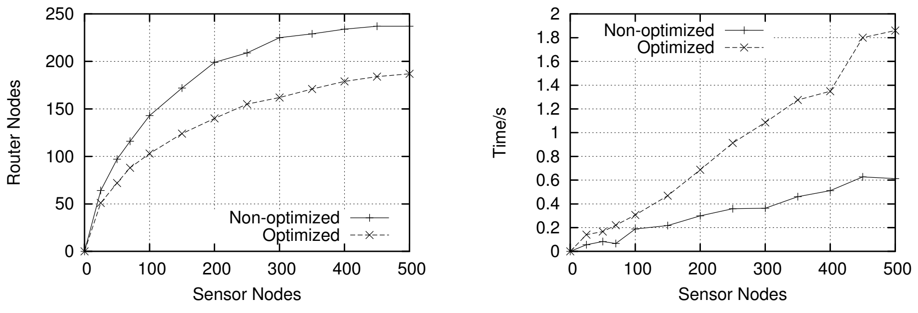


Time used for computation

Figure 5.8: Trivial Reuse Algorithm with Optimization

5.2.3 Cluster Algorithm

The cluster algorithm, with test results in Figure 5.9, has the smallest number of routers removed by the optimization. The time needed for optimization is smaller than for the other algorithms.



Number of router nodes placed Time used for computation

Figure 5.9: Cluster Algorithm with Optimization

5.2.4 Summary

By looking at a comparison of the number of router nodes placed, Figure 5.10, we can see that the “Cluster Algorithm” performs the best when considering the number of router nodes deployed. By looking at this figure we can also see that the number of router nodes seems to asymptotically converge to some value as the number of sensor nodes increase.

Since the number of router nodes in the solutions from the cluster algorithm is lower than the theoretical value obtained earlier, we can draw the conclusion that 500 sensor nodes is not enough to fill the area of deployment.

When considering the time needed to reach a solution we can see that the “Cluster Algorithm” once again is the best choice. A comparison between the different algorithms is shown in Figure 5.11.

In Figure 5.12 the number of router nodes per sensor node is illustrated. We can see that the optimization makes all three algorithms behave similarly with fewer router nodes per sensor node as the number of sensor nodes increase.

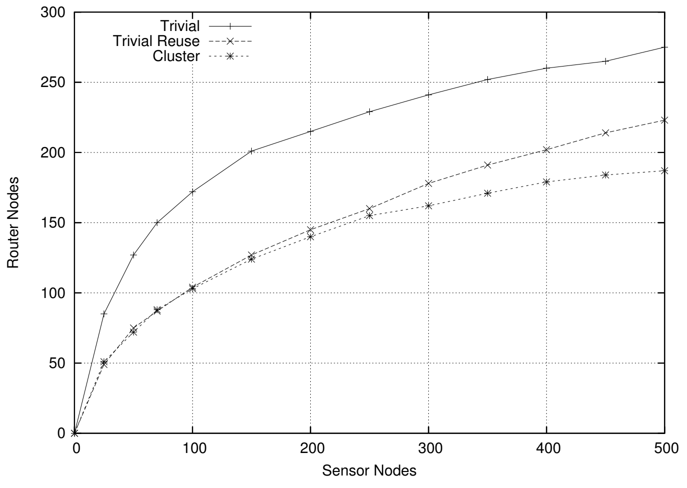


Figure 5.10: Number of router nodes placed by the different algorithms when using optimization.

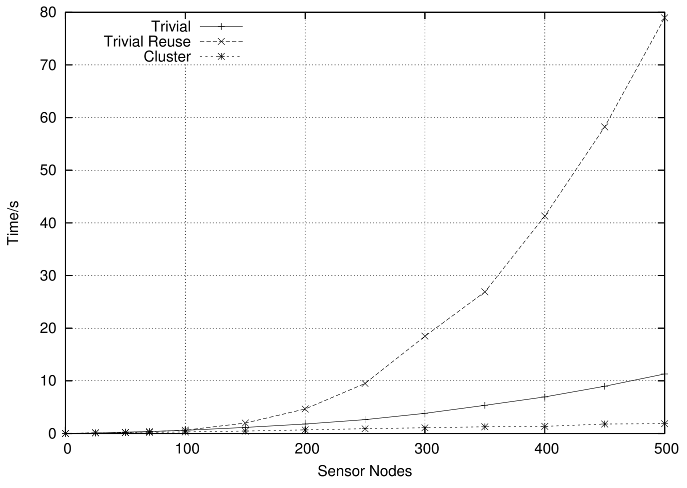


Figure 5.11: Computation time needed by the different algorithms when using optimization.

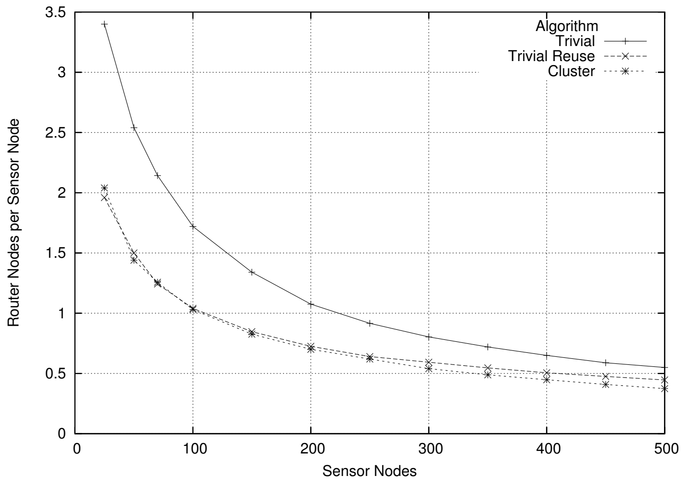
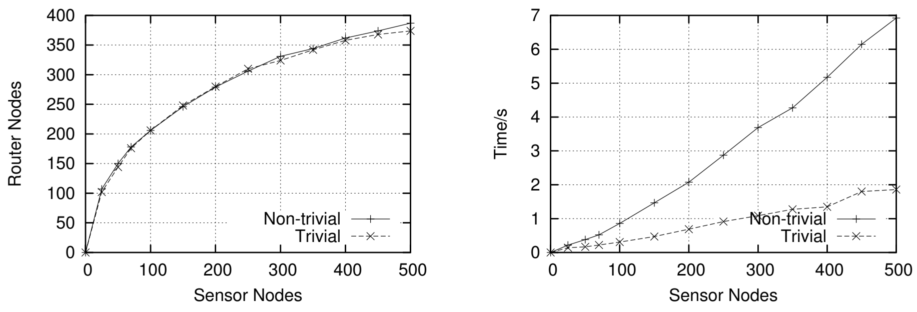


Figure 5.12: Computation time needed by the algorithm with different levels of redundancy and different number of sensor nodes with optimization.

5.3 Redundant Solutions

In this section the redundant algorithm presented in Section 3.5 is evaluated.

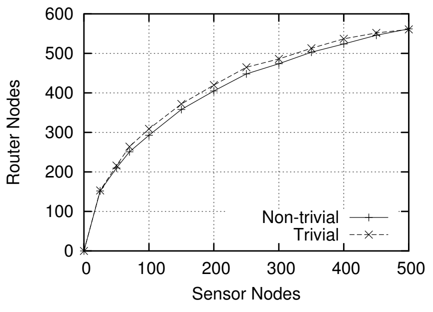
Figure 5.13 shows the number of routers placed in a 2-redundant solution with the non-trivial algorithm compared to the trivial algorithm. It also show the time needed by the non-trivial algorithm and the trivial algorithm. Here we can see that the non-trivial algorithm performs a little bit better than the trivial algorithm but at the cost of longer computation time.



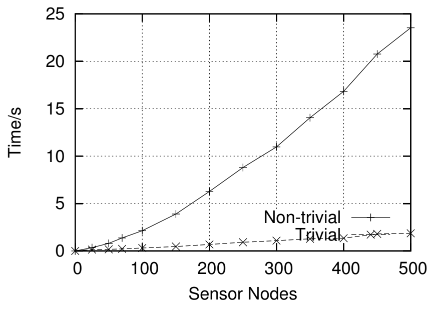
Number of router nodes placed Time used for computation

Figure 5.13: Cluster Algorithm with 2-Redundancy

In Figure 5.14 the results from 3-redundant solutions is shown. The figure shows that the number of router nodes placed is slightly lower in the non-trivial solution than the trivial one and that the time needed for computation is much longer in the non-trivial case than in the trivial case.



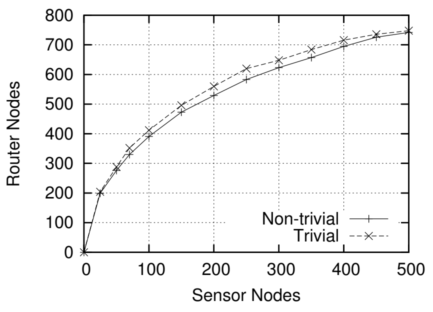
Number of router nodes placed



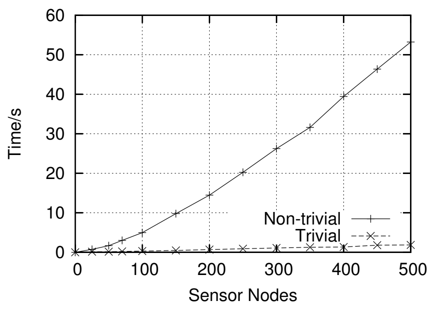
Time used for computation

Figure 5.14: Cluster Algorithm with 3-Redundancy

Finally 4-redundant solutions are considered, the number of router nodes and computation time needed is shown in Figure 5.15. The 4-redundant solutions follow the same pattern as the 2- and 3-redundant solutions, with the number of router nodes placed being only slightly lower in the non-trivial case but at the cost of longer computation time.



Number of router nodes placed



Time used for computation

Figure 5.15: Cluster Algorithm with 4-Redundancy

In Figure 5.16 the number of router nodes placed is plotted against the level of redundancy. In this figure the relationship between redundancy and router nodes placed is visible. We can see that the number of router nodes placed seems to be slightly less than linearly proportional to the level of redundancy in the solution.

Figure 5.17 shows how the time needed for computation increases as the level of redundancy increase. It is clearly visible that the time needed increase steeply as the level of redundancy increase. This is what was predicted in the algorithm analysis in Section 3.5.5.

The number of router nodes per sensor node for different levels of redun-

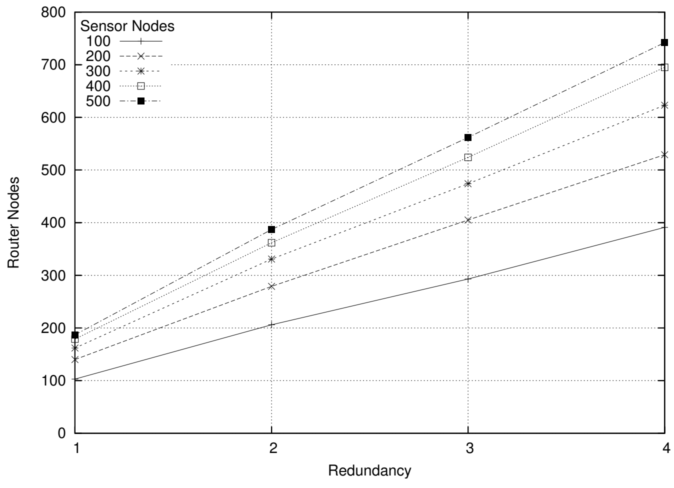


Figure 5.16: Number of router nodes placed by the algorithm with different levels of redundancy and different number of sensor nodes.

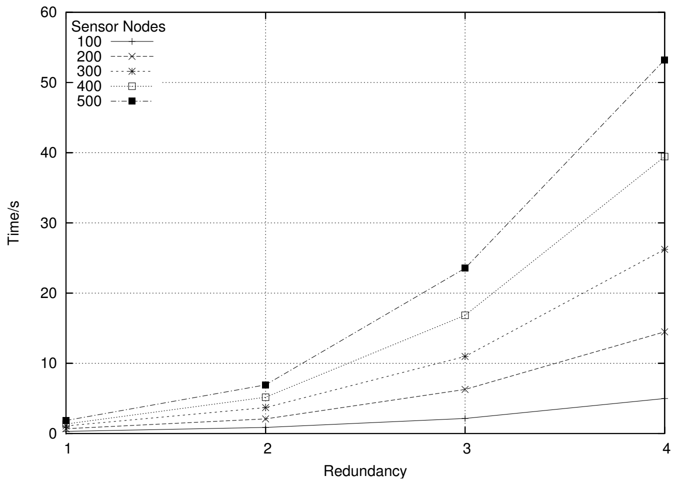


Figure 5.17: Computation time needed by the algorithm with different levels of redundancy and different number of sensor nodes.

dancy is shown in Figure 5.18. It is clearly visible that as the number of sensor nodes increase the number of router nodes per sensor node decrease, the area of deployment becomes saturated with router nodes.

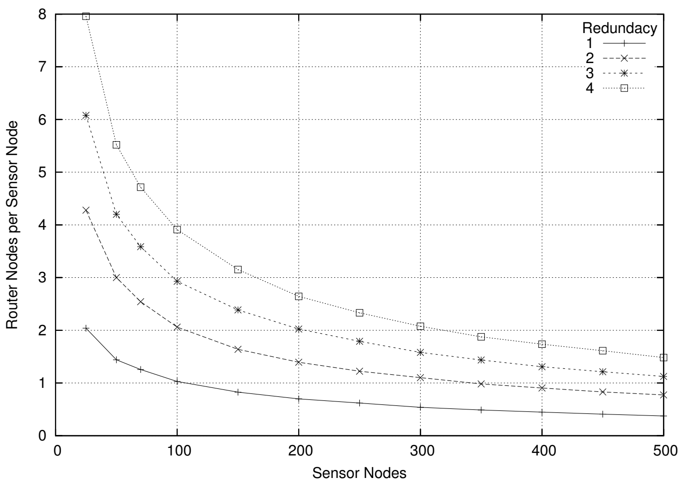


Figure 5.18: Computation time needed by the algorithm with different levels of redundancy and different number of sensor nodes with optimization.

5.4 Redundant Optimized Solutions

In this section the redundant algorithm presented in Section 3.5 optimized by the algorithm discussed in Section 3.6.2 is evaluated.

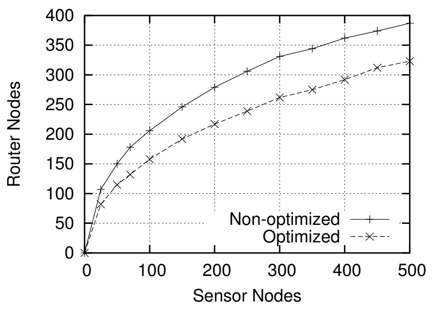
Figure 5.19, 5.20 and 5.21 show the number of router nodes placed and the computation time needed for 2, 3, and 4-redundant solutions respectively.

The figures show that the optimization is able to remove some router nodes but at the cost of much longer computation time.

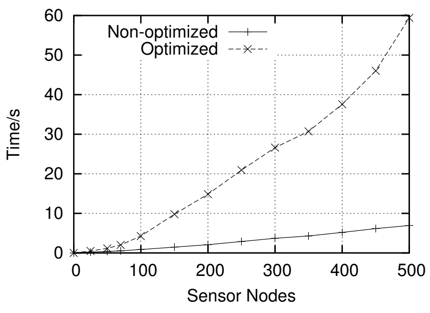
Figure 5.22 shows the number of router nodes plotted against the level of redundancy. We can see that the number of router nodes is close to linearly proportional to the level of redundancy.

The time needed is shown in Figure 5.23. The figure shows that the time needed increase steeply as the level of redundancy increase.

In Figure 5.24 the number of router nodes per sensor node is plotted. The plots are very similar to the non-optimized case with only slightly lower values.

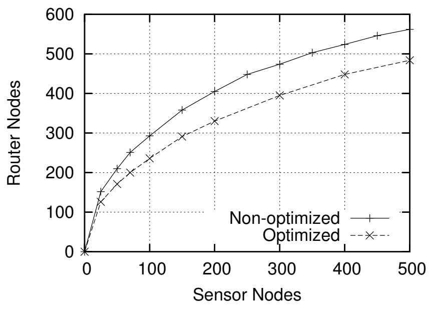


Number of router nodes placed

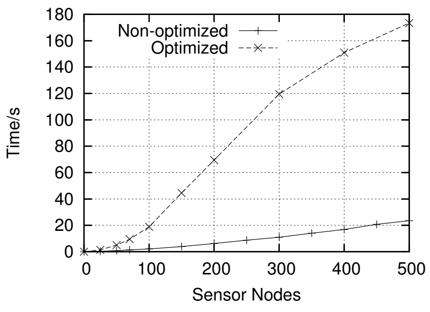


Time used for computation

Figure 5.19: Cluster Algorithm with 2-Redundancy, Optimized

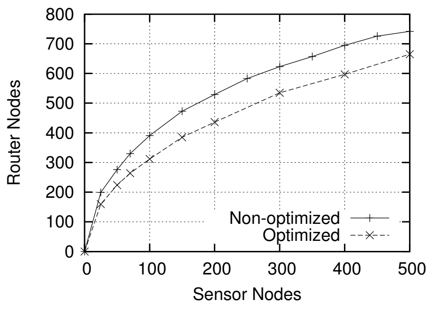


Number of router nodes placed

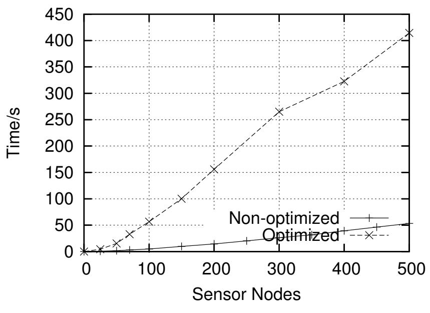


Time used for computation

Figure 5.20: Cluster Algorithm with 3-Redundancy, Optimized



Number of router nodes placed



Time used for computation

Figure 5.21: Cluster Algorithm with 4-Redundancy, Optimized

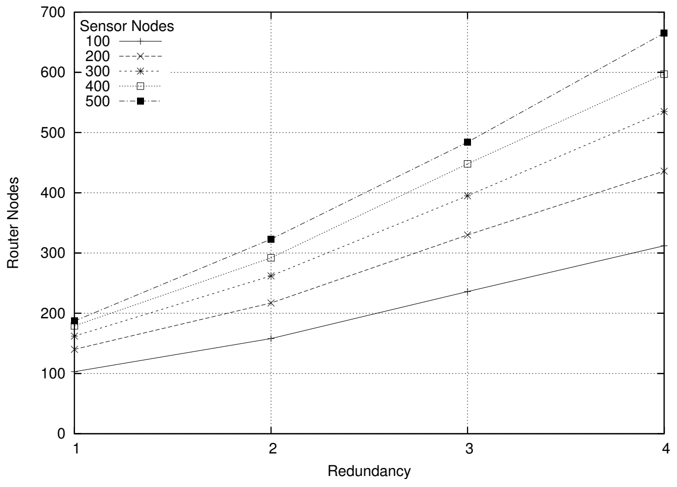


Figure 5.22: Number of router nodes placed by the algorithm with different levels of redundancy and different number of sensor nodes with optimization.

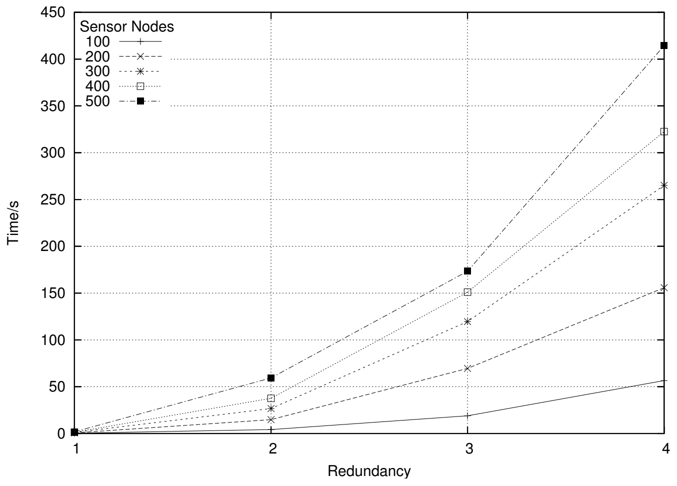


Figure 5.23: Computation time needed by the algorithm with different levels of redundancy and different number of sensor nodes with optimization.

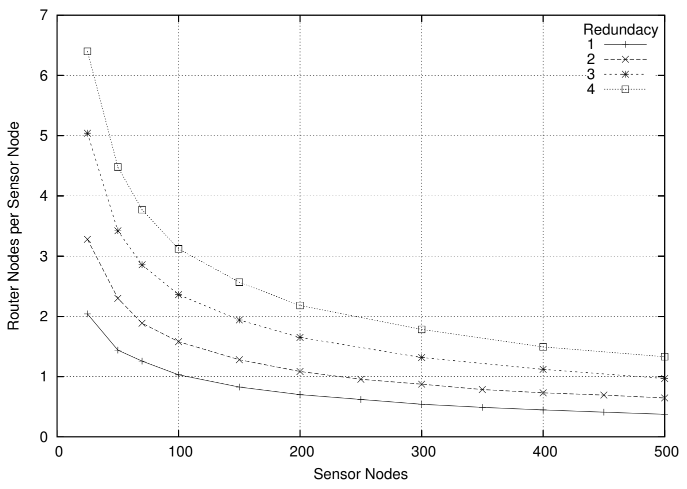


Figure 5.24: Computation time needed by the algorithm with different levels of redundancy and different number of sensor nodes with optimization.

6. Conclusions

6.1 Conclusions

In this thesis different approaches for placement of router nodes in wireless sensor networks have been designed, analyzed, implemented and evaluated.

6.1.1 Design

Three non-redundant algorithms has been designed for this project:

Trivial Places router nodes on straight lines from every sensor node to the gateway.

Trivial Reuse Places router nodes on straight lines from every sensor node to the closest router node.

Cluster Finds clusters of sensor nodes and connects the cluster to the closest router node.

The redundant solutions start with a non-redundant solution and uses an approach similar to the "Trivial Reuse" algorithm in order to create new routes.

To improve the solutions two optimizers have been designed, one that optimizes non-redundant solutions and one that can optimize redundant solutions.

6.1.2 Implementation

The algorithms designed for this project has been implemented in a Java application. Here the algorithms have been tested and the results logged for evaluation purposes.

6.1.3 Evaluation

Non-Redundant Solutions

In the evaluation we could see that we can create non-redundant solutions with a number of router nodes close to the theoretical lower bound of about 226 router nodes by using the algorithm that searches for clusters of sensor nodes and then connects the clusters.

A comparison of the evaluation results (measured time and placed router nodes) for the different non-redundant non-optimized algorithms is shown in Table 6.1.

We can see that the “Cluster” algorithm places the least number of router nodes and that it is quite fast. We can also see that the number of router nodes per sensor node is constant as the number of sensor nodes increase for the “Trivial” algorithm and that it decreases when using the other algorithms.

	200 Sensor Nodes		500 Sensor Nodes	
Algorithm	Router Nodes	Time/s	Router Nodes	Time/s
Trivial	1091	0.069	2713	0.12
Trivial Reuse	248	4.3	458	73
Cluster	199	0.30	237	0.61

Table 6.1: Comparison of non-redundant non-optimized algorithms

In Table 6.2 the non-redundant optimized case is shown. We can see that it is possible to remove quite a few router nodes and that the number of router nodes needed vary less between the different algorithms. In this case the “Cluster” algorithm is faster than all others. In the optimized case all algorithms result in lower number of router nodes per sensor node as the number of sensors increase.

	200 Sensor Nodes		500 Sensor Nodes	
Algorithm	Router Nodes	Time/s	Router Nodes	Time/s
Trivial	215	1.8	275	11
Trivial Reuse	145	4.7	223	79
Cluster	140	0.69	187	1.9

Table 6.2: Comparison of non-redundant optimized algorithms

Redundant Solutions

For the redundant case router nodes can be saved by letting routes go in circles instead of simply placing two or more router nodes at the same position.

Table 6.3 shows how the number of router nodes increase as the level of redundancy increase when using the redundancy algorithm presented in this thesis. In the non-optimized case the number of router nodes saved compared to simply multiply router nodes is quite low.

	200 Sensor Nodes		500 Sensor Nodes	
Redundancy	Router Nodes	Time/s	Router Nodes	Time/s
1	140	0.69	187	1.9
2	279	2.1	387	6.9
3	405	6.3	562	24
4	529	14	742	53

Table 6.3: Comparison of redundant non-optimized solutions

The optimized redundant case is shown in Table 6.4. After optimization the number of router nodes is quite lower than in the trivial case, but at the cost of increased computation time.

	200 Sensor Nodes		500 Sensor Nodes	
Redundancy	Router Nodes	Time/s	Router Nodes	Time/s
1	140	0.69	187	1.9
2	217	14.9	323	59
3	330	69	484	170
4	436	160	665	410

Table 6.4: Comparison of redundant optimized solutions

6.2 Future Work

In this thesis the 2-dimensional solutions have been considered and the 3-dimensional case have been left out for simplicity. The algorithms are extendable into 3-dimensions but with slightly different characteristics. Therefore the 3-dimensional case needs to be analyzed and evaluated.

Power limitations, an important aspect in wireless sensor networks, has not at all been analyzed in this project. In order to know our network we need an assessment of the lifetime or the network and how it behaves as router nodes fail due to power shortage.

Another property not analyzed in this project is the latency, or network propagation delay. This should be analyzed and possibly used to restrict some parameters such as number of hops in the algorithms.

For the placement of router nodes, we need to add a way to restrict placements. In a real world problem we can not place router nodes anywhere, but we are limited to certain positions.

Another issue is the radio propagation which is not uniform in the real world case. The placement algorithms should take this into consideration when placing router nodes.

One possible way to further reduce the number router nodes in a solution is to move router nodes in a completed solution. This should be analyzed further and possibly implemented and evaluated.

References

[1] Bluetooth Special Interest Group. http://www.bluetooth.com/.

[2] Ed Callaway, Paul Gorday, Lance Hester, Jose A. Gutierrez, and Marco Naeve. Home networking with ieee 802.15.4: A developing standard for low-rate wireless personal area networks. *IEEE Communications Magazine*, pages 70–77, August 2002.

[3] Benjamin A. Chambers. The grid roofnet: a rooftop ad hoc wireless network. Master's thesis, MIT, 2002.

[4] Samir R. Das, Charles E. Perkins, and Elizabeth M. Royer. Performance comparision of two on-demand routing protocols for mobile ad hoc networks. *IEEE Personal Communications*, February 2001.

[5] Michael T. Goodrich and Roberto Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons, Inc, 2002.

[6] José A. Gutierrez, Marco Naeve, Ed Callaway, Monique Bourgeois, Vinay Milter, and Bob Heile. Ieee 802.15.4: A developing standard for low-power low-cost wireless personal area networks. *IEEE Network*, pages 12–19, September/October 2001.

[7] Jaap C. Haartsen and Sven Mattisson. Bluetooth—a new low-power radio interface providing short-range connectivity. In *Proceedings of the IEEE, IEEE Special Issue on Low-Power RF Systems*, volume 88, pages 1651–1661, October 2000.

[8] Pai-Hsiang Hsiao and H. T. Kung. Layout design for multiple collocated wireless mesh networks. In *Proceedings of IEEE VTC*, September 2004.

[9] Chi-Fu Huang and Yu-Chee Tseng. The coverage problem in a wireless sensor network. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 115–121, 2003.

[10] IEEE Computer Society. *IEEE Standards 802.15.4 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs).* IEEE, October 2003. ISBN 0-7381-3686-7.

[11] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[12] George Karypis, Euil-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, pages 68–75, August 1999.

[13] Mauricio Marengoni, Bruce Draper, Allen Hanson, and Ramesh Sitaraman. A system to place observers to cover a polyhedral terrain in polybominal time, 1999.

[14] Muriel Médard, Steven G. Finn, Richard A. Barry, and Robert G. Gallager. Redundant trees for preplanned recovery in arbitary vertex-redundant or edge-redundant graphs. *IEE/ACM Transactions on Networking*, 7(5):641–652, October 1999.

[15] Seapahn Meguerdichian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *INFOCOM*, pages 1380–1387, 2001.

[16] C. Perkins. Ad hoc on demand distance vector (aodv) routing, 1997.

[17] Domenico Porcino and Walter Hirt. Ultra-wideband radio technology: potential and challenges ahead. *IEEE Communications Magazine*, pages 66–74, July 2003.

[18] Xiaorui Wang, Guoliang Xing, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. Integrated coverage and connectivity configuration in wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 28–39. ACM Press, 2003.

[19] ZigBee$^{\text{TM}}$Alliance. http://www.zigbee.org/.

Appendix A

Table of Acronyms and Abbreviations

AODR	Ad Hoc On-Demand Distance Vector Routing
DSR	Dynamic Source Routing
FFD	Full-Function Device
IEEE	Institute of Electrical and Electronics Engineers
MAC	Medium Access Control
MANET	Mobile Ad Hoc Networks
RFD	Reduced Function Device
UWB	Ultra Wide Band
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network